

Obtaining Information about Queries Behind Views and Dependencies

Rada Chirkova

Department of Computer Science
NC State University, Raleigh, NC 27695, USA
chirkova@csc.ncsu.edu

Ting Yu

Department of Computer Science
NC State University, Raleigh, NC 27695, USA
yu@csc.ncsu.edu

ABSTRACT

We consider the problems of finding and determining certain query answers and of determining containment between queries; each problem is formulated in presence of materialized views and dependencies under the closed-world assumption. We show a tight relationship between the problems in this setting. Further, we introduce algorithms for solving each problem for those inputs where all the queries and views are conjunctive, and the dependencies are embedded weakly acyclic [13]. We also determine the complexity of each problem under the security-relevant complexity measure introduced in [31]. The problems studied in this paper are fundamental in ensuring correct specification of database access-control policies, in particular in case of fine-grained access control. Our approaches can also be applied in the areas of inference control, secure data publishing, and database auditing.

1. INTRODUCTION

In this paper, we consider the problems of finding and determining certain answers to relational queries, and of containment between relational queries. For the former two problems, we build on the setting of [1], and for the latter – on the setting of [31]; we point out and exploit a tight relationship between the settings. To begin with, in all these settings the set of databases (a.k.a. instances) of interest is not given directly, and is instead specified via a set of “materialized views.” That is, we are given definitions of one or more named queries (*definitions of views*). We are also given a set of answer tuples for each view, that is, each view is *materialized* into a relation. Intuitively, each set MV of materialized views specifies a set of “base instances” I , such that each relation in MV can be obtained as an answer, on the instance I , to the respective view definition. In addition, for a given set of integrity constraints (*dependencies*) on the instances of interest, we deem relevant only those instances that satisfy all the dependencies. In summary, we consider the problems of finding and determining certain query answers and the problem of query containment, each with respect to the sets of base instances specified by a given set of materialized views and a given set of dependencies.

The following motivating example draws on the area of database security called “database-access control” [7].

EXAMPLE 1.1. Suppose a relation Emp stores information about employees of a company, using attributes $Name$, $Dept$ (department), and $Salary$. Two other relations of interest are $HQDept(Dept)$ and $OfficeInHQ$

$(Name, Office)$. The relation $HQDept$ stores the names of the departments that are located in the company headquarters; $OfficeInHQ$ associates employees working in the headquarters with their office addresses.

We now describe the integrity constraint (dependency) that holds on the database schema $P = \{ Emp, HQDept, OfficeInHQ \}$. Suppose that for all the departments located in the company headquarters, all their employees have their offices in the headquarters. This can be expressed as a “tuple-generating dependency” [2], which we call σ . (Please see Example 4.1 for a formalization.)

Let a “secret query” [20] Q ask for the names and salaries of all the employees who work in the company headquarters. We can formulate Q in the standard relational query language SQL , as follows:

```
(Q):SELECT DISTINCT Emp.Name, Salary FROM Emp, OfficeInHQ
WHERE Emp.Name = OfficeInHQ.Name;
```

Consider three views, U , V , and W , that are defined for some class(es) of users, in SQL on the schema P . The view U returns the relation $HQDept$, the view V returns the department names for each employee, and W returns the salaries in each department:

```
(U): DEFINE VIEW U(Dept) AS SELECT * FROM HQDept;
(V): DEFINE VIEW V(Name,Dept) AS
      SELECT DISTINCT Name, Dept FROM Emp;
(W): DEFINE VIEW W(Dept,Salary) AS
      SELECT DISTINCT Dept, Salary FROM Emp;
```

Consider database users who are authorized to see only the answers to the views U , V and W . (In particular, these users are not authorized to see any answers to the query Q .) That is, U , V , and W are access-control views for these users on the database with the schema P . Suppose that at some point in time, these users can see the following set MV of answers to the views:

$MV = \{U(sales), V(johnDoe, sales), W(sales, 50000)\}$.

A basic security question in this setting is as follows: Can these users find out which tuples must be in the answer to the query Q on all the relevant “back-end” databases? If the answer to this question is positive, then, intuitively, there is a security breach, in the form of leakage of some answers to Q to unauthorized users.

Let the back-end databases of interest be those instances of schema P (i.e., “base instances”) that satisfy the dependency σ and that “generate exactly MV as answers to U , V , and W .” (The latter requirement is the “closed-world assumption,” to be discussed in detail shortly.) Using an algorithm introduced in this paper, we can show that the tuple $\bar{t} = (johnDoe, 50000)$ is in

the answer to the secret query Q on all such base instances. Thus, we can answer the above security question in the positive for the secret query Q in the “materialized-view setting” $(P, \{\sigma\}, \{U, V, W\}, MV)$. \square

A tuple that is in the answer to the query of interest on all the relevant base instances is called a *certain answer* to the query. “Determining a certain query answer” is the problem of determining if a given tuple is a certain answer to a given query w.r.t. the given materialized views and, possibly, dependencies. (E.g., the tuple \bar{t} in Example 1.1 is a certain answer to the query Q in the setting $(P, \{\sigma\}, \{U, V, W\}, MV)$.) The problems of finding and determining certain query answers on the instances defined by the given materialized views have been considered both under the “open-world assumption” (OWA) and under the “closed-world assumption” (CWA). That is, for a base instance I , consider the answer tuples generated by the given view definitions on I . Then, informally, I is relevant to the given instance MV of materialized views under CWA iff these answer tuples together comprise exactly MV . In contrast, OWA permits MV to not contain all such tuples.

The classic paper [1] by Abiteboul and Duschka addressed the complexity of determining certain query answers under both OWA and CWA, for a range of query and view languages and in the absence of dependencies. [1] also provided algorithms for finding certain query answers under both OWA and CWA, for queries defined in datalog and for views in nonrecursive datalog, with disequalities (\neq) permitted in both, again in the absence of dependencies. The algorithms of [1] are based on the “conditional tables” of [17]. The formulation of the latter problem was extended, in the context of database security, to account for database dependencies; the extended problem was solved in [8, 27], under OWA for more restricted (than in [1]) languages of queries and views and for “embedded dependencies” [2].

Our original motivation for this current work comes from the fact that finding certain query answers is a basic problem in database security, as illustrated by Example 1.1. Moreover, in its security form, this problem makes the most sense under CWA, rather than under OWA (see, e.g., [24]). Intuitively, for those database attackers who are seeking unauthorized answers to a secret query, Q , in presence of a set of view answers MV , the only relevant base instances are those that “generate exactly MV ,” that is only the CWA-relevant instances. Suppose the owners of the back-end database run an OWA-based algorithm for finding certain answers to Q w.r.t. MV . They could then arrive at the empty set of answers (and thus conclude that their database is secure), even though under CWA, the set of certain answers to Q would not be empty for the same MV and dependencies. Indeed, we can use the results of [8, 27] to show that in Example 1.1, the set of certain answers to the query Q is the empty set under OWA.¹

To address this challenge, we have developed a CWA-based approach for finding certain answers to conjunctive queries (*CQ queries*), in presence of CQ views and of weakly acyclic embedded dependencies [13]. (Similarly to the dependency-free case [1], the CWA version of this problem is harder than the OWA version consid-

ered in [8, 27].) We then realized that our techniques can be connected to the solution of [31], by Zhang and Mendelzon, to the (CWA-based) problem of determining containment between queries in presence of materialized views. The latter problem arises, for instance, in determining whether a user query formulated on the base database relations has an equivalent rewriting in terms of the access-control views for this user. (If the answer to the question is positive, then the user query can be answered safely, see [24, 31] for the details.) A natural and practically important generalization of this problem is its extension to the consideration of dependencies holding on the base instances. We have been able to extend to the case of dependencies the algorithm of [31] for their query-containment problem w.r.t. materialized views MV , by building on our approach to the problem of finding certain query answers w.r.t. MV .

Our contributions.

Our specific contributions are as follows:

- We formalize the problem of determining containment between two queries under CWA and in presence of materialized views and dependencies, by building on the formalization of [31] that does not consider dependencies.
- We develop an algorithm for solving this problem, in the case where the input queries and all the view definitions are CQ queries, and the input dependencies are embedded weakly acyclic.
- We show that the problem of determining certain answers to a query, under CWA and in presence of materialized CQ views and dependencies, is a special case of the above containment problem. It follows that the algorithm that we introduce for the containment problem also solves correctly this “certain-query-answer” problem, for all inputs where the queries and views are CQ queries and the dependencies are embedded weakly acyclic.
- For the problem of *finding* all certain query answers under CWA and in presence of materialized views and dependencies, we develop two algorithms that are sound and complete for all inputs where the queries and views are CQ queries and the dependencies are embedded weakly acyclic. The first algorithm uses as a subroutine our algorithm for the “certain-query-answer” problem. The second algorithm both builds on the standard approach to answering queries in relational data exchange, and uses a simpler version of the technique that we use to solve the above *containment* problem (w.r.t. materialized views and dependencies).
- We determine the complexity of each of the three problems under the security-relevant complexity measure of [31]. In this measure, it is assumed that everything is fixed except for the materialized views and queries (but not for view definitions).

The problems that we study in the paper can be used to model and analyze a wide range of database-security problems, including database-policy analysis, secure data publishing, inference control, and auditing. For instance, database-security policies are often implemented through views. It is important to ensure that security views are defined correctly, so that no sensitive information can be learned by unauthorized par-

¹The algorithms of [1] could not be applied to the problem instance of Example 1.1, as that instance contains a dependency that is not a “full” [2] (a.k.a. “total” [6]) dependency.

ties from granted view access [8, 29]. Clearly, an information disclosure happens if an attacker can learn certain answers to a secret query. The same modeling can be applied to capture the secure data-publishing problem [20]. Similarly, in database query auditing, answers to user-issued queries can be modeled as materialized views [22, 23]. A potential inference attack happens if those answers combined together can be used to derive secret information as defined by a query.

The remainder of the paper is organized as follows. After discussing related work in Section 2, we review the background definitions and results in Section 3, and then define our three problems of interest in Section 4. In Sections 5–6 we introduce our approaches to solving the three problems in the CQ weakly acyclic setting. Finally, in Section 7 we address the complexity of the three problems in the CQ weakly acyclic setting.

2. RELATED WORK

The seminal paper [1] by Abiteboul and Duschka addressed the complexity of the problem of determining whether a tuple is a certain query answer in presence of materialized views (the “certain-query-answer problem”) under both OWA and CWA, for a range of query and view languages and in the absence of dependencies. [1] also solved the problem of finding all certain query answers under both OWA and CWA, for datalog queries and for views in nonrecursive datalog, with disequalities (\neq) permitted in both, in the absence of dependencies. The algorithms of [1] are based on the “conditional tables” of [17]. (See [2, 16] for detailed overviews of incomplete databases and of their representations, including conditional tables [17].) It is remarked in [1] that their algorithms for finding certain answers could be extended to the case of “full” (or “total”) dependencies [6, 2]. In this current paper, we provide sound and complete algorithms for finding certain answers and for the certain-query-answer problem, under CWA for CQ queries and views and for “weakly acyclic” [13] embedded dependencies, of which the class of full dependencies is a proper subclass. We also address the complexity of both problems in this CQ weakly acyclic setting.

The paper [8] by Brodsky and colleagues introduced in the security context the problem of finding certain query answers under OWA and in presence of embedded dependencies, and proposed a sound and complete algorithm for the case where the queries and views are CQ queries expressible without joins. Then, Stoffel and colleagues in [27] made a connection between this problem and the techniques introduced in data exchange [13, 5, 4], by developing ([27]) a data-exchange-based approach for finding certain query answers, under OWA for CQ views, UCQ queries (i.e., unions of CQ queries), and embedded dependencies.

In this paper we extend the data-exchange approach of [13], also used in [27], to solve the problem of finding certain query answers under CWA, for CQ queries and views in presence of weakly acyclic embedded dependencies. The approaches of this current paper do not use “target-to-source dependencies” introduced in the data-exchange context in [15]. The dependencies in our approaches do use constants (as was suggested back in [13]), and thus are related to “conditional dependencies” [14]. Conditional dependencies are intuitively understood as enforcing a (perhaps constant-involving) pattern onto (typically constant-determined) subsets of

the given relations. As the dependencies that we use do not have constants in their antecedents, they intuitively behave in the ways expected of standard (constant-free) embedded dependencies.

While the term “data exchange” is mentioned in the paper [20] by Miklau and Suciu, data-exchange methods are not used in the technical development in [20]. Rather, the term is used in [20] informally as a reference to today’s universal sharing of data (as in, e.g., on the Web). [20] addresses the problem of “data publishing,” in which the goal is to determine, for a given set of view definitions and for a “secret query” Q , whether any materializations of the given views would disclose information about any answers to Q . (In contrast, in the three problems considered in this current paper, we assume that a specific set of view materializations is provided in the problem input.) Further, the notion of disclosure in [20], inspired by Shannon’s notion of perfect secrecy [25], is as follows: There is no disclosure of query Q via views \mathcal{V} if and only if the *probability* of an adversary guessing the answer to Q is the same (or, in another scenario, is almost the same) whether the adversary knows the answers to \mathcal{V} or not. In this current paper, we use a deterministic, rather than probabilistic, notion of disclosure of a query answer, in presence of a specific set of view materializations; this leads to different security decisions than those following from [20].

The work [31] by Zhang and Mendelzon introduced and solved the problem of “conditional containment” between two CQ queries in presence of materialized CQ views, under CWA and in the absence of dependencies. [31] also introduced a security-relevant complexity metric, under which their problem is Π_2^P complete. ([31] also provides an excellent overview of the connections of the query-containment problem of [31] to database-theory literature.) In our work, we add dependencies to the formulation of the problem of [31], and extend the approach of [31], both to solve the resulting problem in the CQ weakly acyclic setting and to analyze the complexity of the problem. We also uncover a tight relationship of the problem with the problems of finding and determining certain query answers, under CWA in presence of view materializations and of dependencies.

3. PRELIMINARIES

3.1 Instances and Queries

Schemas and instances. A *schema* \mathbf{P} is a finite sequence $\langle P_1, \dots, P_m \rangle$ of relation symbols, with each P_i having a fixed arity $k_i \geq 0$. An *instance* I of \mathbf{P} assigns to each $P_i \in \mathbf{P}$ a finite k_i -ary relation $I[P_i]$, which is a set of tuples. For tuple membership in relation $I[P_i]$, we use the notation $\bar{t} \in I[P_i]$. Each element of each tuple in an instance belongs to one of two disjoint infinite sets of values, CONST and VAR. We call elements of CONST *constants*, and denote them by lowercase letters a, b, c, \dots ; the elements of VAR are called (*labeled*) *nulls*, denoted by symbols $\perp, \perp_1, \perp_2, \dots$.

Sometimes we use the notation $P_i(\bar{t}) \in I$ instead of $\bar{t} \in I[P_i]$, and call $P_i(\bar{t})$ a *fact* of I . When all the values in \bar{t} are constants, we say that $P_i(\bar{t})$ is a *ground fact*, and \bar{t} is a *ground tuple*. The *active domain* of instance I , denoted $adom(I)$, is the set of all the elements of $\text{CONST} \cup \text{VAR}$ that occur in any facts in I . When each fact in I is a ground fact, we call I a *ground instance*.

Queries. We consider the class of queries called

“unions of conjunctive queries with disequalities,” UCQ^\neq queries. In the definitions for UCQ^\neq queries, we will use the following notions of *relational atom* and of *(dis)equality atom*. Let QVAR be an infinite set of values disjoint from $\text{CONST} \cup \text{VAR}$; we call QVAR the set of (query) variables. We will denote variables by uppercase letters X, Y, \dots . Then $P(\bar{t})$, with P a k -ary relation symbol and \bar{t} a k -vector of values, is a *relational atom* whenever each value in \bar{t} is an element of $\text{CONST} \cup \text{QVAR}$. Further, an *equality* (resp. *disequality*) atom is a built-in predicate of the form $S \theta T$, where θ is = (resp. \neq), and each of S and T is an element of $\text{CONST} \cup \text{QVAR}$.

A CQ^\neq -rule over schema \mathbf{P} , with k -ary ($k \geq 0$) output relation symbol $Q \notin \mathbf{P}$, is an expression of the form

$$Q(\bar{X}) \leftarrow P_1(\bar{U}_1) \wedge \dots \wedge P_n(\bar{U}_n) \wedge C.$$

Here, $n \geq 1$; the vector \bar{X} has k elements; for each $i \in [1, n]$, $P_i \in \mathbf{P}$; each of $Q(\bar{X})$, $P_1(\bar{U}_1)$, \dots , $P_n(\bar{U}_n)$ is a relational atom; and C is a (possibly empty) finite conjunction of disequality atoms. We consider only *safe* rules: That is, each variable in \bar{X} , as well as each variable occurring in C , also occurs in at least one of $\bar{U}_1, \dots, \bar{U}_n$. All the variables of the rule that do not appear in \bar{X} (i.e., the *nonhead variables* of the rule) are assumed to be existentially quantified. We call the atom $Q(\bar{X})$ the *head* of the rule, call \bar{X} the *head vector* of the rule, and call the conjunction of its remaining atoms the *body* of the rule. Each atom in the body of a rule is called a *subgoal* of the rule. The conjunction in the body is usually written using commas, as $P_1(\bar{U}_1), \dots, P_n(\bar{U}_n), C$.

A *conjunctive query with disequalities* (a CQ^\neq query) is a query defined by a single CQ^\neq -rule; a *conjunctive query* (a CQ query) is a CQ^\neq query with an empty C . We will be referring to a CQ^\neq query with head $Q(\bar{X})$ as just $Q(\bar{X})$, or even Q , whenever clear from the context. We will be using body_Q as a concise name for the body of the (rule for) Q .

Finally, for a k -ary relation symbol Q , with $k \geq 0$, let $\mathcal{S}(Q) = \{ Q^{(1)}, \dots, Q^{(l)} \}$ be a finite set of CQ^\neq -rules over schema \mathbf{P} , such that Q is the output relation symbol in each rule. Then we say that the set $\mathcal{S}(Q)$ defines a UCQ^\neq query Q over \mathbf{P} , and that each element of $\mathcal{S}(Q)$ defines a CQ^\neq component of Q . In the special case where $\mathcal{S}(Q) = \emptyset$, we say that the corresponding UCQ^\neq query Q is a *trivial query*.

Semantics of UCQ^\neq queries. We now define the semantics of a UCQ^\neq query Q . In the definition, we will need the notions of *homomorphism* and of *valuation*. Consider two conjunctions, $\varphi(\bar{Y})$ and $\psi(\bar{Z})$, of relational atoms. Then a mapping h from the set of elements of \bar{Y} to the set of elements of \bar{Z} is called a *homomorphism* from $\varphi(\bar{Y})$ to $\psi(\bar{Z})$ whenever (i) $h(c) = c$ for each constant c in \bar{Y} , and (ii) for each conjunct of the form $p(\bar{U})$ in $\varphi(\bar{Y})$, the relational atom $p(h(\bar{U}))$ is a conjunct in $\psi(\bar{Z})$. (For a vector $\bar{S} = [s_1 s_2 \dots s_l]$, for some $l \geq 0$, we define $h(\bar{S})$ as the vector $[h(s_1)h(s_2)\dots h(s_l)]$. By convention, a homomorphism is an identity mapping when applied to empty vectors and to empty tuples.)

We define homomorphisms in the same way for the case where either one of $\varphi(\bar{Y})$ and $\psi(\bar{Z})$ (or both) is a conjunction of facts. Further, for a conjunction C of disequality atoms and two conjunctions $\varphi(\bar{Y})$ and $\psi(\bar{Z})$ of

relational atoms or of facts, we say that every homomorphism, h , from $\varphi(\bar{Y})$ to $\psi(\bar{Z})$ is also a homomorphism from $\varphi(\bar{Y})$ to $\psi(\bar{Z}) \wedge C$. We will denote homomorphisms by lowercase letters g, h, \dots , possibly with subscripts.

Now suppose we are given a conjunction $\varphi(\bar{Y})$ of relational atoms, a conjunction $\psi(\bar{Z})$ of facts, and a conjunction C of disequalities over variables in \bar{Y} and constants in CONST . Suppose there is a homomorphism, h , from $\varphi(\bar{Y})$ to $\psi(\bar{Z})$, such that for each atom of the form $S \neq T$ in C , the values $h(S)$ and $h(T)$ are distinct elements of $\text{CONST} \cup \text{VAR}$. Then we say that h is a *valuation* from $\varphi(\bar{Y}) \wedge C$ to $\psi(\bar{Z})$. We will use Greek letters μ, ν, \dots , possibly with subscripts, for valuations.

Given a k -ary CQ^\neq query $Q(\bar{X})$ and given an instance I , which we interpret as a conjunction of all the facts in I . Then the answer to Q on I , denoted $Q(I)$, is

$$Q(I) = \{ \nu(\bar{X}) \mid \nu \text{ is a valuation from } \text{body}_Q \text{ to } I \}.$$

(When $k = 0$, i.e., Q is a *Boolean query*, $\nu(\bar{X})$ is the empty tuple.) Further, for a UCQ^\neq query Q defined by $l \geq 1$ rules $\{ Q^{(1)}, \dots, Q^{(l)} \}$, and for an instance I , the answer to Q on I is the union $\cup_{i=1}^l Q^{(i)}(I)$. By convention, for every trivial UCQ^\neq query Q and for every instance I , we have $Q(I) = \emptyset$.

Query containment. A query Q_1 is contained in query Q_2 , denoted $Q_1 \sqsubseteq Q_2$, if $Q_1(I) \subseteq Q_2(I)$ for every instance I . A classic result in [9] by Chandra and Merlin states that a necessary and sufficient condition for the containment $Q_1 \sqsubseteq Q_2$, for CQ queries Q_1 and Q_2 of the same arity, is the existence of a containment mapping from Q_2 to Q_1 . Here, a *containment mapping* [9] from CQ query $Q_2(\bar{X}_2)$ to CQ query $Q_1(\bar{X}_1)$ is a homomorphism h from body_{Q_2} to body_{Q_1} such that $h(\bar{X}_2) = \bar{X}_1$. By the results in [19], this containment test of [9] remains true when Q_1 has built-in predicates. Thus, the same test holds in particular when Q_1 is a CQ^\neq query. It follows that, for Q_1 a UCQ^\neq query and for Q_2 a CQ query, determining whether $Q_1 \sqsubseteq Q_2$ is decidable. Indeed, the containment holds iff for each CQ^\neq rule $Q_1^{(i)} \in \mathcal{S}(Q_1)$, $i \in [1, l]$, we have $Q_1^{(i)} \sqsubseteq Q_2$.

Canonical database. Every CQ^\neq query Q can be regarded as a symbolic ground instance $I^{(Q)}$. $I^{(Q)}$ is defined as the result of turning each relational atom $P_i(\dots)$ in body_Q into a tuple in the relation $I^{(Q)}[P_i]$. The procedure is to keep each constant in the body of Q , and to replace consistently each variable in the body of Q by a distinct constant different from all the constants in Q . The tuples that correspond to the resulting ground facts are the only tuples in the *canonical database* $I^{(Q)}$ for Q , which is unique up to isomorphism.

Remark. We have defined CQ^\neq -rules as not having explicit equality atoms in their bodies. As a result and by definition of canonical database, we are restricting our consideration to the set of all and only *satisfiable* CQ^\neq -rules/queries. (A CQ^\neq -rule/query Q is *satisfiable* iff there exists an instance I such that $Q(I) \neq \emptyset$.)

3.2 Dependencies and Chase

Embedded dependencies. We consider dependencies σ of the form

$$\sigma : \varphi(\bar{U}, \bar{V}) \rightarrow \exists \bar{W} \psi(\bar{U}, \bar{W}) \quad (1)$$

with ϕ and ψ conjunctions of relational atoms, possibly with equations added. (All the variables in \bar{U} , \bar{V} are understood to be universally quantified.) Such dependencies, called *embedded dependencies*, are expressive enough to specify all usual integrity constraints, such as keys, foreign keys, and inclusion dependencies [2]. If ψ is a single equation, then σ is an *equality-generating dependency* (egd). If ψ consists only of relational atoms, then σ is a *tuple-generating dependency* (tgd). We follow [13] in allowing constants in egds and tgds. Each set of embedded dependencies without constants is equivalent to a set of tgds and egds [2]. We write $I \models \Sigma$ if instance I satisfies all elements of set Σ of dependencies. All the sets Σ that we refer to are finite.

Query containment under dependencies. We say that query Q is contained in query P under set of dependencies Σ , denoted $Q \sqsubseteq_{\Sigma} P$, if for every instance $I \models \Sigma$ we have $Q(I) \subseteq P(I)$. Queries Q and P are equivalent under Σ , denoted $Q \equiv_{\Sigma} P$, if both $Q \sqsubseteq_{\Sigma} P$ and $P \sqsubseteq_{\Sigma} Q$ hold. Q and P are equivalent (in the absence of dependencies), denoted $Q \equiv P$, if $Q \equiv_{\emptyset} P$.

Chase for CQ queries. Given a CQ query $Q(\bar{X}) \leftarrow \xi(\bar{X}, \bar{Y})$ and a tgd σ as in Eq. (1); assume w.l.o.g. that Q has none of the variables in \bar{W} . The (standard [16]) *chase of Q with σ is applicable* if there is a homomorphism h from ϕ to ξ , such that h cannot be extended to a homomorphism from $\phi \wedge \psi$ to ξ . Then, a (standard) *chase step* on Q with σ and h is a rewrite of Q into a CQ query $Q^*(\bar{X}) \leftarrow \xi(\bar{X}, \bar{Y}) \wedge \psi(h(\bar{U}), \bar{W})$. It can be shown that $Q^* \equiv_{\{\sigma\}} Q$ and that $Q^* \sqsubseteq Q$.

We now define a (standard [16]) *chase step* with an egd. Assume a CQ query Q , as before, and an egd σ of the form $\phi(\bar{U}) \rightarrow U_1 = U_2$. The *chase of Q with σ is applicable* if there is a homomorphism h from ϕ to ξ such that $h(U_1) \neq h(U_2)$. Suppose at least one of $h(U_1)$ and $h(U_2)$ is a variable; let w.l.o.g. $h(U_1)$ be a variable. Then a *chase step* on Q with σ and h is a rewrite of Q into a CQ query, Q^* , that results from replacing all occurrences of $h(U_1)$ in Q by $h(U_2)$. Again, we have $Q^* \equiv_{\{\sigma\}} Q$ and $Q^* \sqsubseteq Q$. If, for an h as above, $h(U_1)$ and $h(U_2)$ are distinct constants, then we say that *chase with σ fails on Q* . In this case, $Q(I) = \emptyset$ on all $I \models \{\sigma\}$.

A Σ -chase sequence \mathcal{C} (or just *chase sequence*, if Σ is clear from the context) for CQ query Q_0 is a sequence of CQ queries Q_0, Q_1, \dots such that each query Q_{i+1} ($i \geq 0$) in \mathcal{C} is obtained from Q_i by a chase step $Q_i \Rightarrow^{\sigma} Q_{i+1}$ using a dependency $\sigma \in \Sigma$. A chase sequence $Q = Q_0, Q_1, \dots, Q_n$ is *terminating* if $I^{(Q_n)} \models \Sigma$, where $I^{(Q_n)}$ is the canonical database for Q_n . In this case we denote Q_n by $(Q)^{\Sigma}$ and say that $(Q)^{\Sigma}$ is the (terminal) *result* of the chase. All chase results for a given CQ query are equivalent in the absence of dependencies [11].

Weakly acyclic dependencies [13]. Let Σ be a set of tgds over schema \mathbf{T} . We construct the *dependency graph* of Σ , as follows. The nodes (positions) of the graph are all pairs (T, A) , for $T \in \mathbf{T}$ and A an attribute of T . We now add edges: For each tgd $\varphi(\bar{X}) \rightarrow \exists \bar{Y} \psi(\bar{X}, \bar{Y})$ in Σ , and for each $X \in \bar{X}$ that occurs in φ in position (T, A) and that occurs in ψ , do the following.

- For each occurrence of X in ψ in position (S, B) , add a *regular edge* from (T, A) to (S, B) ; and
- For each existentially quantified variable $Y \in \bar{Y}$ and for each occurrence of Y in ψ in position $(R,$

$C)$, add a *special edge* from (T, A) to (R, C) .

For a set Σ of tgds and egds, with Σ^t the set of all tgds in Σ , we say that Σ is *weakly acyclic* if the dependency graph of Σ^t does not have a cycle going through a special edge. Chase of CQ queries terminates in finite time under sets of weakly acyclic dependencies [13].

The following result is immediate from [2, 10, 11, 18].

THEOREM 3.1. *Given CQ queries Q_1, Q_2 and a set Σ of embedded dependencies. Then $Q_1 \sqsubseteq_{\Sigma} Q_2$ iff $(Q_1)^{\Sigma} \sqsubseteq Q_2$ in the absence of dependencies.* \square

Chase of instance. Let I be an instance of schema \mathbf{P} , and Σ a set of egds and tgds; we interpret I as a conjunction of its facts. We follow [11] in defining chase of I with Σ in the same way as chase of a CQ query with Σ . That is, in the chase steps we treat each distinct null in I as a distinct variable (in the chase for CQ queries). Further, each chase step with a tgd that has existential variables introduces, in the result of the chase step, a distinct new null for each existential variable of the tgd. Chase sequences and chase termination are also defined in the same way as for CQ queries; the result I' of the chase of I with Σ always satisfies Σ , that is, $I' \models \Sigma$.

4. THE PROBLEM STATEMENTS

In this section we formalize the problems of finding and determining certain query answers and of query containment, under CWA and in presence of dependencies. We then establish a direct relationship between the latter two problems in the case of CQ view definitions.

4.1 Certain Query Answers and Query Containment w.r.t. Views and Dependencies

We begin by introducing the notion of “materialized-view setting” (“setting” for short). Suppose that we are given a schema \mathbf{P} and a set of dependencies Σ on \mathbf{P} . Let \mathcal{V} be a finite set of relation symbols not in \mathbf{P} , with each symbol (*view name*) $V \in \mathcal{V}$ of some arity $k_V \geq 0$. Each $V \in \mathcal{V}$ is associated with a k_V -ary query on the schema \mathbf{P} . We call \mathcal{V} a *set of views on \mathbf{P}* , and call the query for each $V \in \mathcal{V}$ the *definition of the view V* , or the *query for V* . We assume that the query for each $V \in \mathcal{V}$ is associated with $(V \text{ in } \mathcal{V})$ the set \mathcal{V} . We call a ground instance MV of schema \mathcal{V} a *set of view answers for \mathcal{V}* .

Let I be a ground instance of schema \mathbf{P} . We say that I is a Σ -valid base instance for \mathcal{V} and MV , denoted by $\mathcal{V} \Rightarrow_{I, \Sigma} MV$, whenever (a) $I \models \Sigma$, and (b) the answer $V(I)$ to the query for V on the instance I is identical to the relation $MV[V]$ in MV , for each $V \in \mathcal{V}$. (This is the *closed-world assumption (CWA)*, as defined in, e.g., [1], with an added requirement that $I \models \Sigma$.) Further, we say that MV is a Σ -valid set of view answers for \mathcal{V} , denoted by $\mathcal{V} \Rightarrow_{*, \Sigma} MV$, whenever there exists a Σ -valid base instance for \mathcal{V} and MV .

DEFINITION 4.1. (Materialized-view setting $\mathcal{M}\Sigma$) *Given a schema \mathbf{P} , a set Σ of dependencies without constants on \mathbf{P} , a set \mathcal{V} of views on \mathbf{P} , and a $(\Sigma$ -valid) set MV of view answers for \mathcal{V} : We call $\mathcal{M}\Sigma = (\mathbf{P}, \Sigma, \mathcal{V}, MV)$ the (valid) materialized-view setting for \mathbf{P} , Σ , \mathcal{V} , and MV .* \square

Let $\mathcal{M}\Sigma$ be a materialized-view setting $(\mathbf{P}, \Sigma, \mathcal{V}, MV)$, and let Q be a query over \mathbf{P} . We define the *set of certain answers of Q w.r.t. the setting $\mathcal{M}\Sigma$* as

$\text{certain}_{\mathcal{M}\Sigma}(Q) = \bigcap \{Q(I) \mid I \text{ s.t. } \mathcal{V} \Rightarrow_{I,\Sigma} MV \text{ in } \mathcal{M}\Sigma\}.$

That is, the set of certain answers of a query w.r.t. a setting is understood, as usual, as the set of all tuples that are in the answer to the query on all the instances relevant to the setting. (Cf. [1] for the case $\Sigma = \emptyset$.)

DEFINITION 4.2. (Certain-query-answer problem in a materialized-view setting) *Given a setting $\mathcal{M}\Sigma = (\mathbf{P}, \Sigma, \mathcal{V}, MV)$, a k -ary ($k \geq 0$) query Q over the schema \mathbf{P} in $\mathcal{M}\Sigma$, and a ground k -tuple \bar{t} . Then the certain-query-answer problem for Q and \bar{t} in $\mathcal{M}\Sigma$ is to determine whether $\bar{t} \in \text{certain}_{\mathcal{M}\Sigma}(Q)$. \square*

It is easy to show that a tuple \bar{t} can be a certain answer to a query Q in a setting $\mathcal{M}\Sigma$ only if all the values in \bar{t} are in $\text{consts}(\mathcal{M}\Sigma)$, which denotes the set of constants occurring in $\mathcal{M}\Sigma$. (For a given materialized-view setting $\mathcal{M}\Sigma = (\mathbf{P}, \Sigma, \mathcal{V}, MV)$, we define $\text{consts}(\mathcal{M}\Sigma)$ as the union of $\text{atom}(MV)$ with the set of all the constants used in the definitions of the views \mathcal{V} .) By this observation, in Definition 4.2 we can restrict our consideration to the tuples \bar{t} with this property.

The problem as in Definition 4.2, the *problem of determining certain query answers*, will be featured in our characteristic of the relationship between the extensions of the problems of [1] and of [31] to the case of dependencies under CWA. We will also consider the *problem of finding the set of certain query answers w.r.t. a setting*: Given a setting $\mathcal{M}\Sigma$ and a query Q , find the set of certain answers of Q w.r.t. $\mathcal{M}\Sigma$. In Sections 5–6, we will introduce algorithms for solving the “CQ weakly acyclic case” of this problem and of the problem of Definition 4.2. The *CQ weakly acyclic case* of each problem is the case where: (i) each $\mathcal{M}\Sigma$ is *conjunctive* (i.e., all the views in $\mathcal{M}\Sigma$ are defined as CQ queries) and *weakly acyclic* (i.e., Σ in $\mathcal{M}\Sigma$ is a set of weakly acyclic embedded dependencies), and (ii) each Q is a CQ query.

We now turn our attention to the problem of query containment w.r.t. a setting $\mathcal{M}\Sigma$. Our Definition 4.3 extends the formalization of this problem due to [31], to the case of dependencies on the relevant base instances.

DEFINITION 4.3. ($\mathcal{M}\Sigma$ -conditional query containment) *Given a materialized-view setting $\mathcal{M}\Sigma$ and queries Q_1 and Q_2 over the schema \mathbf{P} in $\mathcal{M}\Sigma$. Then we say that Q_1 is $\mathcal{M}\Sigma$ -conditionally contained in Q_2 , denoted $Q_1 \sqsubseteq_{\mathcal{M}\Sigma} Q_2$, iff for each instance I s.t. $\mathcal{V} \Rightarrow_{I,\Sigma} MV$ in $\mathcal{M}\Sigma$, we have $Q_1(I) \subseteq Q_2(I)$. Further, the problem of $\mathcal{M}\Sigma$ -conditional containment for Q_1 and Q_2 is to determine whether $Q_1 \sqsubseteq_{\mathcal{M}\Sigma} Q_2$. \square*

4.2 An Illustration

In this subsection we recast Example 1.1 into the formal terms of Section 4.1. The results of this paper permit us to obtain correct solutions to all the three problems formulated at the end of Example 4.1.

EXAMPLE 4.1. *The setting $\mathcal{M}\Sigma$ outlined in Example 1.1 uses the schema² $\mathbf{P} = \{E, H, O\}$ and a weakly acyclic set $\{\sigma\}$ of dependencies, with σ as follows:*

$$\sigma: E(X, Y, Z) \wedge H(Y) \rightarrow \exists S \ O(X, S).$$

Further, $\mathcal{V} = \{U, V, W\}$ is the set of CQ views in $\mathcal{M}\Sigma$, with the view definitions as follows:

²We abbreviate the relation names of Example 1.1 using the first letter of each name.

$$\begin{aligned} U(X) &\leftarrow H(X). \\ V(X, Y) &\leftarrow E(X, Y, Z). \\ W(Y, Z) &\leftarrow E(X, Y, Z). \end{aligned}$$

Finally, for brevity we encode the constants of Example 1.1 as c for johnDoe, d for sales, and f for 50000. Then the set of view answers MV of Example 1.1 can be recast for $\mathcal{M}\Sigma$ as $MV = \{U(d), V(c, d), W(d, f)\}$.

Now that we have specified a CQ weakly acyclic setting $\mathcal{M}\Sigma$, consider the CQ query Q of Example 1.1:

$$Q(X, Z) \leftarrow E(X, Y, Z), O(X, S).$$

Consider another CQ query, Q_1 , defined as follows:

$$Q_1(c, f) \leftarrow H(d), E(c, d, X), E(Y, d, f).$$

For the $\mathcal{M}\Sigma$, Q , and Q_1 as above and for a tuple $\bar{t} = (c, f)$, we have the following problems as in Section 4.1:

1. *The certain-query-answer problem for Q and \bar{t} in $\mathcal{M}\Sigma$ is “Is \bar{t} a certain answer of Q w.r.t. $\mathcal{M}\Sigma$?”*
2. *The problem of finding the set of certain answers to Q w.r.t. $\mathcal{M}\Sigma$ is “Return the set $\text{certain}_{\mathcal{M}\Sigma}(Q)$ for Q and $\mathcal{M}\Sigma$ ”; and, finally,*
3. *The problem of $\mathcal{M}\Sigma$ -conditional containment for Q_1 and Q is “Does $Q_1 \sqsubseteq_{\mathcal{M}\Sigma} Q$ hold?” \square*

4.3 Relationship between the Problems

We now establish a direct relationship between the certain-query-answer problem for a given Q , \bar{t} , and $\mathcal{M}\Sigma$, and the problem of $\mathcal{M}\Sigma$ -conditional containment for Q_1 and Q , for the same Q and $\mathcal{M}\Sigma$. (We prove the relationship for the case where all the views are defined as CQ queries.) Here, the query Q_1 is constructed from the given Q , \bar{t} , and $\mathcal{M}\Sigma$. A similar relationship was observed in [1] between the certain-query-answer problem, for a range of query and view languages in the dependency-free case under OWA, and *unconditional* ($Q_1 \sqsubseteq Q$) query containment. In contrast, our result holds under CWA, in presence of dependencies, and involves $\mathcal{M}\Sigma$ -conditional query containment. Due to this result, the algorithm that we introduce in Section 5 for checking $\mathcal{M}\Sigma$ -conditional containment, can also be used to solve the certain-query-answer problem, in the CQ weakly acyclic case of each problem. (The *CQ weakly acyclic case* of the containment problem covers CQ weakly acyclic settings and CQ input queries.)

We formulate the main result of this section, Theorem 4.1, using the following notation. For a set $\mathcal{V} = \{V_1, \dots, V_m\}$ of $m \geq 1$ CQ views and for a set MV of view answers for \mathcal{V} , consider the conjunction

$$\mathcal{C}_{MV} = \bigwedge_{i=1}^m \bigwedge_{j=1}^{l_i} V_i(\bar{t}_{ij})$$

The conjunction is over all the ground facts $V_i(\bar{t}_{ij})$ in the set MV . (For each $i \in [1, m]$, the relation $MV[V_i]$ in MV is of cardinality $l_i \geq 0$.) That is, we treat each ground fact in MV as a relational atom, and \mathcal{C}_{MV} is the conjunction of all these relational atoms. (For each i such that $MV[V_i] = \emptyset$, we define $\bigwedge_{j=1}^{l_i} V_i(\bar{t}_{ij}) := \text{true}$.)

Observe that \mathcal{C}_{MV} can be treated as the body of a CQ query over the schema \mathcal{V} . Thus, we can use the view definitions in \mathcal{V} to do the standard *expansion* (as in a rewriting [19]) of \mathcal{C}_{MV} into a conjunction of atoms, \mathcal{C}_{MV}^{exp} , over the schema \mathbf{P} . We call \mathcal{C}_{MV}^{exp} the *expansion*

of MV over \mathbf{P} . As an illustration, in the setting of Example 4.1, \mathcal{C}_{MV} is $U(d) \wedge V(c, d) \wedge W(d, f)$, and \mathcal{C}_{MV}^{exp} is the body of the query Q_1 in the example.

We now formulate Theorem 4.1. (Due to the page limit, the straightforward proof and other details can be found in Appendix B.) This result says that for a valid CQ materialized-view setting $\mathcal{M}\Sigma$ and for an arbitrary query Q and an arbitrary ground tuple \bar{t} , there exists a (constructible) CQ query Q_1 such that the certain-query-answer problem for Q and \bar{t} in $\mathcal{M}\Sigma$ is the problem of $\mathcal{M}\Sigma$ -conditional containment for Q_1 and Q .

THEOREM 4.1. *Given a valid CQ materialized-view setting $\mathcal{M}\Sigma = (\mathbf{P}, \Sigma, \mathcal{V}, MV)$, a k -ary ($k \geq 0$) query Q defined in an arbitrary query language over \mathbf{P} , and a k -tuple \bar{t} of values in $\text{consts}(\mathcal{M}\Sigma)$. Consider the CQ query $Q_1(\bar{t}) \leftarrow \mathcal{C}_{MV}^{exp}$. Then $\bar{t} \in \text{certain}_{\mathcal{M}\Sigma}(Q)$ if and only if Q_1 is $\mathcal{M}\Sigma$ -conditionally contained in Q . \square*

Whenever determining validity of a setting $\mathcal{M}\Sigma$ is decidable (as is the case for, e.g., CQ weakly acyclic settings, via our view-verified data-exchange approach of Section 6, see Appendix J.3), $\mathcal{M}\Sigma$ not being valid implies that $\text{certain}_{\mathcal{M}\Sigma}(Q) = \emptyset$ for every query Q .

5. THE QUERY-CONTAINMENT PROBLEM

In this section we outline our approach to solving the problem of $\mathcal{M}\Sigma$ -conditional query containment. (See Definition 4.3.) We show that this approach is a correct algorithm for the CQ weakly acyclic case of the problem. Thus, our algorithm extends to the case of weakly acyclic dependencies the solution of [31] for their problem of conditional containment between CQ queries in presence of materialized CQ views.³ We show that our extension of the method of [31] is not trivial. By Theorem 4.1, the approach reported in this section is also a correct algorithm for the CQ weakly acyclic cases of the certain-query-answer problem.

5.1 Intuition and Discussion

We begin by sketching our containment-checking approach via an extended example. The example illustrates, in particular, how disequalities and disjunction may arise in the chase of a CQ query in this approach.

EXAMPLE 5.1. *Consider CQ queries Q_1 and Q_2 :*

$$\begin{aligned} Q_1(X) &\leftarrow P(X, Y). \\ Q_2(X) &\leftarrow P(X, Y), R(Z). \end{aligned}$$

Consider a dependency (full tgd) σ on the schema $\mathbf{P} = \{P, R\}$, a view V , and an instance MV , as follows.

$$\begin{aligned} \sigma : P(X, X) &\rightarrow R(X) \\ V(X) &\leftarrow P(X, X). \\ MV &= \{V(c)\}. \end{aligned}$$

Let us specify a setting $\mathcal{M}\Sigma$ as $(\mathbf{P}, \{\sigma\}, \{V\}, MV)$. The setting $\mathcal{M}\Sigma$ is CQ weakly acyclic by definition.

By the results reviewed in Section 3, the query Q_1 is not unconditionally contained in Q_2 , either in the absence of dependencies or in presence of σ . At the same time, by our results in this section, Q_1 is $\mathcal{M}\Sigma$ -contained in Q_2 . Our approach to proving it is by chasing the query Q_1 using a “ \neq -transformation,” $\sigma_{(\neq)}$, of the given

tgd σ on the schema \mathbf{P} , as well as “MV-induced dependencies.” (We introduce both kinds of dependencies in Section 5.2.) The first step of the approach is to conjoin the body of Q_1 with $\mathcal{C}_{MV}^{exp} = P(c, c)$ (see Section 4.3 for the definition of \mathcal{C}_{MV}^{exp}):

$$Q'_1(X) \leftarrow P(X, Y), P(c, c).$$

Now the only MV-induced dependency, τ_V , is

$$\tau_V : P(X, Y) \rightarrow (X = c \wedge Y = c) \vee (X \neq Y).$$

It says that, for each subgoal of the form $P(X, Y)$ that could arise in the chase of Q'_1 with the dependencies τ_V and $\sigma_{(\neq)}$: Either (i) the subgoal must become $P(c, c)$, which would (correctly) give rise to $V(c)$ in MV , or (ii) $P(X, Y)$ must be accompanied by the disequality $X \neq Y$, to prevent atoms of the form $V(d)$, where d is a constant not equal to c , from arising in MV . (These requirements must be satisfied for our approach to be correct, see Proposition 5.3 in Section 5.3.)

The chase of Q'_1 with τ_V produces a UCQ $^\neq$ query:

$$\begin{aligned} Q_1^{(a)}(c) &\leftarrow P(c, c), P(c, c). \text{ (We then drop the duplicate.)} \\ Q_1^{(b)}(X) &\leftarrow P(X, Y), P(c, c), X \neq Y. \end{aligned}$$

Now the dependency $\sigma_{(\neq)}$, which we obtain from the tgd σ , is $\sigma_{(\neq)} : P(X, Y) \rightarrow R(X) \vee X \neq Y$. Applying $\sigma_{(\neq)}$ to the above UCQ $^\neq$ query yields the UCQ $^\neq$ result $(Q_1)^{\mathcal{M}\Sigma} = \{Q_1^{(1)}, Q_1^{(2)}, Q_1^{(3)}\}$ of chasing the query Q'_1 with the dependencies τ_V and $\sigma_{(\neq)}$:

$$\begin{aligned} Q_1^{(1)}(c) &\leftarrow P(c, c), R(c). \\ Q_1^{(2)}(X) &\leftarrow P(X, Y), R(X), P(c, c), R(c). \\ Q_1^{(3)}(X) &\leftarrow P(X, Y), X \neq Y, P(c, c), R(c). \end{aligned}$$

Now the results of [19] can be used to ascertain the unconditional containment of $(Q_1)^{\mathcal{M}\Sigma}$ in the query Q_2 . We conclude that the query Q_1 is $\mathcal{M}\Sigma$ -contained in Q_2 .

Finally, suppose that we change the query Q_2 slightly, by replacing its subgoal $R(Z)$ with $R(X)$. Then the same procedure as above can be used to show that the resulting query would not $\mathcal{M}\Sigma$ -contain the query Q_1 . \square

In some particularly simple cases, queries $(Q_1)^{\mathcal{M}\Sigma}$ can be CQ queries; see Appendix F. In general in our approach, queries $(Q_1)^{\mathcal{M}\Sigma}$ are UCQ $^\neq$ queries.

In our proposed approach for checking $\mathcal{M}\Sigma$ -containment of CQ queries, the intuition is the same as in checking query containment in presence of dependencies [2, 10, 11, 18] (see Section 3). That is, to determine if a query Q_1 is contained in query Q_2 on a set of instances that are “relevant” to a set of view answers MV , we chase Q_1 to transform it into a query, $(Q_1)^{\mathcal{M}\Sigma}$, which is equivalent, by construction, to Q_1 on all the relevant instances. (The “relevant instances” are the Σ -valid base instances for the given \mathcal{V} and MV .) In addition to this property, the query $(Q_1)^{\mathcal{M}\Sigma}$, by its construction, “exhibits the flavor of the relevant instances,” in a very precise sense (see Proposition 5.3 in Section 5.3). These properties permit us to use a test for unconditional containment of $(Q_1)^{\mathcal{M}\Sigma}$ in Q_2 to correctly determine whether the original query Q_1 is contained in Q_2 w.r.t. all the relevant instances. (See Theorem 5.1 in Section 5.3.)

³A full version of [31], including proofs of its results, has never been published.

Zhang and Mendelzon in their paper [31] did precisely the above chase, with precisely the same goals and results, in the special case where no dependencies hold on the relevant instances. As an illustration, suppose that in Example 5.1 we set $\Sigma := \emptyset$, while keeping the remaining inputs as they are. Then the approach of [31] for these inputs would derive the UCQ^\neq query $\{Q_1^{(a)}, Q_1^{(b)}\}$ of that example, call this query Q_1'' . As Q_1'' is not unconditionally contained in the given query Q_2 , the conclusion of [31] for these inputs would be that Q_2 does not contain Q_1 w.r.t. these inputs with $\Sigma = \emptyset$.

Thus, in this current work we build directly on the ideas and techniques of [31]. At the same time, [31] does not make the chase process explicit, in the way in which it is explicit in the work (e.g., [2, 10, 11, 18]) on determining containment of queries in presence of dependencies. In particular, the paper [31] does not introduce dependencies that look like τ_V in Example 5.1. As a result, the authors of [31] do not have to deal with the (arguably inelegant) extensions of embedded dependencies to dependencies that may have disjunction and disequalities on the right-hand side. (Appendix C provides some details of the approach of [31].)

In this current paper, when extending the approach of [31] to the case of dependencies holding on the instances of interest, it has proved convenient for us to make explicit the MV -induced dependencies, such as τ_V in Example 5.1. Thus, in this work we introduce (in Section 5.2) dependencies that have both disjunctions and disequalities on the right-hand side. Disequalities in dependencies are necessary in our approach for determining $\mathcal{M}\Sigma$ -conditional containment, see Section 5.3. (As a side note, we will see in Section 6 that disequalities in dependencies are *not* necessary for essentially the same approach to work correctly when solving the problem of finding the set of certain answers to a CQ query w.r.t. a CQ weakly acyclic materialized-view setting.)

Not surprisingly, for CQ weakly acyclic settings $\mathcal{M}\Sigma$ and CQ queries Q_1 and Q_2 of interest, $Q_1 \sqsubseteq_{\mathcal{M}\Sigma} Q_2$ does not necessarily imply any of the following:

- $Q_1 \sqsubseteq Q_2$;
- $Q_1 \sqsubseteq_\Sigma Q_2$; and
- $Q_1 \sqsubseteq_{\mathcal{M}\emptyset} Q_2$; here, by $\mathcal{M}\emptyset$ we denote the result of replacing Σ by \emptyset in $\mathcal{M}\Sigma$.

(See Appendix E for all the details.)

5.2 The Dependencies and Chase Rules

We now introduce dependencies that are used in the algorithm of Section 5.3. The input to each run of the algorithm is a triple of the form $(\mathcal{M}\Sigma, Q_1, Q_2)$, with $\mathcal{M}\Sigma$ a CQ weakly acyclic setting, and Q_1 and Q_2 two CQ queries. We call such triples *CQ weakly acyclic input instances*. For each $(\mathcal{M}\Sigma, Q_1, Q_2)$, the algorithm determines whether $Q_1 \sqsubseteq_{\mathcal{M}\Sigma} Q_2$ holds. To make the determination, a modification (via adding \mathcal{C}_{MV}^{exp}) of the query Q_1 is chased with the dependencies that we introduce in the current subsection.

Building blocks for the chase.

All the dependencies used in Section 5.3 are constructed using the input CQ setting $\mathcal{M}\Sigma$. (For ease of exposition, in the remainder of this subsection we will assume that one such setting $\mathcal{M}\Sigma = (\mathbf{P}, \Sigma, \mathcal{V}, MV)$ is fixed.) The construction uses *normalized* versions of

conjunctions of relational atoms (see, e.g., [30]). That is, let ϕ be a conjunction of relational atoms. We replace in ϕ each duplicate occurrence of a variable or constant with a fresh distinct variable name. As we do each replacement, say of X (or c) with Y , we add to the conjunction the equality atom $Y = X$ (or $Y = c$). As an illustration, if $\phi = P(X, X) \wedge S(c, c, X)$, then its normalized version is $\phi^{(n)} = P(X, Y) \wedge S(c, Z, W) \wedge Y = X \wedge Z = c \wedge W = X$. By construction, the normalized version of each ϕ is unique up to variable renamings. For the normalized version $\phi^{(n)}$ of a conjunction ϕ , we will denote by $\mathcal{R}(\phi^{(n)})$ the conjunction of all the relational atoms in $\phi^{(n)}$, and will denote by $\mathcal{E}(\phi^{(n)})$ the conjunction of all the equality atoms in $\phi^{(n)}$. (If $\phi^{(n)}$ has no equality atoms, we set $\mathcal{E}(\phi^{(n)})$ to *true*.)

A *non-egd* (*negd*) is a dependency of the form

$$\sigma : \phi(\bar{W}) \rightarrow X \neq Y. \quad (2)$$

Here, ϕ is a conjunction of relational atoms, and each of X and Y is an element of the set of variables \bar{W} .

We also use chase with “implication constraints,” see, e.g., [30]. An *implication constraint* (*ic*) is a dependency of the form $\tau : \phi(\bar{W}) \rightarrow \text{false}$, with $\phi(\bar{W})$ a conjunction of relational atoms.

The algorithm of Section 5.3 performs chase of CQ^\neq queries with ics, negds, egds, and tgds, by the following rules. Let Q be a CQ^\neq query. We say that *chase of Q with an ic τ is applicable* whenever there exists a homomorphism, h , from the antecedent ϕ of τ to the body of Q . Then we say that the *chase step of Q with τ fails*. Similarly, we say that a *chase step with a negd σ (as in Eq. (2)) applies to Q* if there exists a homomorphism, h , from the antecedent ϕ of σ to the body of Q . There are two cases: One, $h(X)$ and $h(Y)$ are the same variable (or the same constant) in Q . Then we say that *the chase step of Q with σ fails*. Otherwise, we form from Q the *result* Q^* of the chase step: Q^* is a CQ^\neq query obtained by conjoining $\text{body}_{(Q)}$ with the atom $h(X) \neq h(Y)$. Chase steps with *tgds* are defined for CQ^\neq queries in the same way as for CQ queries, see Section 3.2. Finally, for chase with *egds*, we extend the rules of Section 3.2 by requiring that whenever chase of a CQ^\neq query Q with an egd τ is applicable, with some homomorphism h , and the consequent of τ is of the form $X = Y$, then *the chase step of Q with τ fails* iff $\text{body}_{(Q)}$ has the atom $h(X) \neq h(Y)$ (or $h(Y) \neq h(X)$). (This generalizes the chase-step rule for CQ queries with egds, in the part where $h(X)$ and $h(Y)$ are distinct constants, see Section 3.2.) As we define CQ^\neq queries as not having explicit equality atoms, our extended chase-step rules cover all possible cases for CQ^\neq queries.

Dependencies $\Phi_{(MV)}$ for CQ setting $\mathcal{M}\Sigma$.

We now introduce one type of dependencies, *MV-induced dependencies* $\Phi_{(MV)}$, to be used in the chase in the algorithm of Section 5.3. For the CQ setting $\mathcal{M}\Sigma$ with set \mathcal{V} of views, let $V \in \mathcal{V}$ be a k_V -ary ($k_V \geq 0$) view with definition $V(\bar{X}) \leftarrow \phi(\bar{X}, \bar{Y})$. We first normalize the body ϕ of V into $\mathcal{R}(\phi^{(n)}) \wedge \mathcal{E}(\phi^{(n)})$. The result $\neg \mathcal{E}(\phi^{(n)})$ of negating $\mathcal{E}(\phi^{(n)})$ is (obviously) a disjunction of disequality atoms. (E.g., $\neg (X = Y \wedge Z = c)$ is $(X \neq Y \vee Z \neq c)$.) We now proceed for V as follows.

If $MV[V] = \emptyset$, we define the *MV-induced generalized implication constraint (MV-induced gic)* ι_V for V as

$$\iota_V : \mathcal{R}(\phi^{(n)}) \rightarrow \text{false} \vee \neg \mathcal{E}(\phi^{(n)}). \quad (3)$$

Now suppose $k_V \geq 1$ and $MV[V] = \{\bar{t}_1, \bar{t}_2, \dots, \bar{t}_{m_V}\}$, with $m_V \geq 1$. Then we define the *MV-induced generalized negd (MV-induced gnegd)* τ_V for V as

$$\tau_V : \mathcal{R}(\phi^{(n)}) \rightarrow \bigvee_{i=1}^{m_V} (\bar{X} = \bar{t}_i) \vee \neg \mathcal{E}(\phi^{(n)}). \quad (4)$$

Here, $\bar{X} = [S_1, \dots, S_{k_V}]$ is the head vector of the query for V , with $S_j \in \text{CONST} \cup \text{QVAR}$ for $j \in [1, k_V]$. (By definition of $\mathcal{R}(\phi^{(n)})$, all the elements of \bar{X} occur in $\mathcal{R}(\phi^{(n)})$.) For each $i \in [1, m_V]$ and for the ground tuple $\bar{t}_i = (c_{i1}, \dots, c_{ik_V}) \in MV[V]$, we abbreviate by $\bar{X} = \bar{t}_i$ the conjunction $\bigwedge_{j=1}^{k_V} (S_j = c_{ij})$. *MV-induced gnegds* are a straightforward generalization of disjunctive egds of [12, 13], with negds added “on top.”

For a CQ setting $\mathcal{M}\Sigma$ with set of view answers MV , the set of *MV-induced dependencies* $\Phi_{(MV)}$ for $\mathcal{M}\Sigma$ is the set of *MV-induced gnegds* and *MV-induced gics* constructed for all the views in $\mathcal{M}\Sigma$ as specified above.⁴

Dependencies $\Sigma_{(\neq)}$ for CQ setting $\mathcal{M}\Sigma$.

We now outline how to obtain from the given CQ setting $\mathcal{M}\Sigma$ the second set of dependencies, $\Sigma_{(\neq)}$, to be used in chase in the algorithm of Section 5.3. We convert each dependency in Σ (in the given $\mathcal{M}\Sigma$) using a conversion rule that follows, and then produce $\Sigma_{(\neq)}$ as the union of the outputs. The conversion rule for a dependency $\sigma \in \Sigma$ of the form $\sigma : \phi(\bar{X}, \bar{Y}) \rightarrow \exists \bar{Z} \psi(\bar{X}, \bar{Z})$ converts ϕ into $\mathcal{R}(\phi^{(n)}) \wedge \mathcal{E}(\phi^{(n)})$, and then returns

$$\sigma_{(\neq)} : \mathcal{R}(\phi^{(n)}) \rightarrow \exists \bar{Z} \psi(\bar{X}, \bar{Z}) \vee \neg \mathcal{E}(\phi^{(n)}).$$

Chase of CQ^\neq queries with $\Upsilon_{\mathcal{M}\Sigma} = \Phi_{(MV)} \cup \Sigma_{(\neq)}$.

We now define chase of CQ^\neq queries with the dependencies $\Upsilon_{\mathcal{M}\Sigma} = \Phi_{(MV)} \cup \Sigma_{(\neq)}$. For the fixed $\mathcal{M}\Sigma$, let Q be a CQ^\neq query over the schema \mathbf{P} in $\mathcal{M}\Sigma$. Our definition of the chase steps can be seen as an extension of the definition of [13] for their disjunctive egds, once we postulate that chase steps are to be applied to queries, rather than to instances as is done in [13]. Intuitively, we view each dependency $v \in \Upsilon_{\mathcal{M}\Sigma}$, of the form $v : \phi \rightarrow \psi_1 \vee \psi_2 \vee \dots \vee \psi_m$, where each ψ_i is a conjunction, as m dependencies $v_1 : \phi \rightarrow \psi_1; \dots; v_m : \phi \rightarrow \psi_m$. Suppose there is a homomorphism, h , from the antecedent ϕ of v to the query Q , and none of $h(\psi_1), h(\psi_2), \dots, h(\psi_m)$ is a tautology. Then we say that the chase step with v applies to Q , and we output, as the result of the step, a set of CQ^\neq queries such that each element of the set results from the application to Q of one of v_1, \dots, v_m , as defined above.

Whenever the chase step of Q with v_i , for an $i \in [1, m]$, fails (as is, e.g., always the case with an ic), then the chase step does not contribute anything to the output set. Thus, if the chase step of Q fails with v_i for *all* $i \in [1, m]$, the output of the chase step of Q with the (original) v is the empty set, i.e., a trivial UCQ^\neq query.

⁴We have shown that it is not necessary to use *MV-induced dependencies* for Boolean views V with $MV[V] \neq \emptyset$.

Once we have a formalization of chase steps of CQ^\neq queries with dependencies $\Upsilon_{\mathcal{M}\Sigma}$, we can define *chase trees* and *chase results*, by generalizing the formalizations of [13] of chase of instances with disjunctive egds. Due to the space limit, we are unable to provide detailed formalizations in the main text. (Example 5.1 provides an illustration. Appendix J has a detailed formalization of the special case where $\Upsilon_{\mathcal{M}\Sigma}$ does not contain any disequalities; an extension to disequalities is straightforward.) Intuitively, in a *chase tree* \mathcal{T} constructed for a CQ setting $\mathcal{M}\Sigma$ and a CQ^\neq query Q , the root represents Q , and each node represents either a CQ^\neq query or (as a special case of a leaf) a trivial UCQ^\neq query; we denote the node in this special case by ϵ . A node t in \mathcal{T} has children t_1, \dots, t_k iff a chase step with some $\sigma \in \Upsilon_{\mathcal{M}\Sigma}$ applies to the CQ^\neq query represented by t , and the result of the chase step is exactly all the queries represented by t_1, \dots, t_k . A (non- ϵ) node t in \mathcal{T} is a leaf iff no dependency in $\Upsilon_{\mathcal{M}\Sigma}$ applies in a chase step to the CQ^\neq query represented by t .

Each chase tree \mathcal{T} can be associated with a sequence (with repeated entries allowed) of dependencies in $\Upsilon_{\mathcal{M}\Sigma}$, according to the sequence of chase steps represented by \mathcal{T} from the root downwards. The *result of the chase of Q with sequence $\sigma_1, \sigma_2, \dots$ of dependencies in $\Upsilon_{\mathcal{M}\Sigma}$* is defined iff the associated \mathcal{T} is a finite tree; then this result is either a trivial UCQ^\neq query (iff each leaf of \mathcal{T} is ϵ), or is the union of all the CQ^\neq queries represented by the leaves of \mathcal{T} . A *chase result of Q with $\mathcal{M}\Sigma$* , denoted $(Q)^{\mathcal{M}\Sigma}$, is the result (if defined) of the chase of Q with any sequence of dependencies in $\Upsilon_{\mathcal{M}\Sigma}$.

We now obtain the following result, in Proposition 5.1, for the case where $\mathcal{M}\Sigma$ is CQ weakly acyclic and Q is a CQ query. Let MV be the set of materialized views in $\mathcal{M}\Sigma$; then \mathcal{C}_{MV}^{exp} is defined as in Section 4.3. As is done in [31], we denote by Q' the CQ query obtained from Q by conjoining the body of Q with \mathcal{C}_{MV}^{exp} , after all the variables of \mathcal{C}_{MV}^{exp} have been consistently renamed so that Q and \mathcal{C}_{MV}^{exp} do not share any variable names. We call Q' the *$\mathcal{M}\Sigma$ -expansion of Q* .

PROPOSITION 5.1. *Given a CQ weakly acyclic setting $\mathcal{M}\Sigma$ and a CQ query Q : For the $\mathcal{M}\Sigma$ -expansion Q' of Q , each chase tree \mathcal{T} for $\mathcal{M}\Sigma$ and Q' is finite, of polynomial depth in the size of Q and of MV in $\mathcal{M}\Sigma$. Further, for any such \mathcal{T} and for the UCQ^\neq query $(Q)^{\mathcal{M}\Sigma}$ that is the result of the chase of Q' with sequence of dependencies associated with \mathcal{T} , we have that:*

- The number of CQ^\neq components of $(Q)^{\mathcal{M}\Sigma}$ is up to exponential in the size of Q and MV , and
- For each CQ^\neq component, q , of $(Q)^{\mathcal{M}\Sigma}$, the size of q is polynomial in the size of Q and MV . \square

The proof of Proposition 5.1 is based on the results of [13], which construct a polynomial-size upper bound on the number of distinct values that can occur in chase of an instance with weakly acyclic tgds and egds. Appendix J outlines a proof for a generalization over [13], in which a version of $\Upsilon_{\mathcal{M}\Sigma}$ is constructed without disequalities; the main observation is that Q' already has all the constants that might be introduced in the chase by the *MV-induced gnegds* (as in Eq. (4)) of $\Upsilon_{\mathcal{M}\Sigma}$. We then build on that result of Appendix J, by observing that chase steps with *negds* do not add new values, and

may add a number of disequality atoms that is only up to polynomial in the size of the given Q and MV .

5.3 The Containment-Checking Algorithm

By Proposition 5.1, if a triple $(\mathcal{M}\Sigma, Q_1, Q_2)$ is CQ weakly acyclic as defined in Section 5.2, then each chase tree for $\mathcal{M}\Sigma$ and Q_1' is finite. Thus, the following procedure, given here by pseudocode, is an algorithm for CQ weakly acyclic inputs. (Testing whether $(\mathcal{M}\Sigma, Q_1, Q_2)$ is CQ weakly acyclic can be done in polynomial time.)

Algorithm $\mathcal{M}\Sigma$ -CONTAINMENT DETERMINATION:

Input: CQ weakly acyclic instance $(\mathcal{M}\Sigma, Q_1, Q_2)$.

Output: Determination whether $Q_1 \sqsubseteq_{\mathcal{M}\Sigma} Q_2$.

1. Set Q_1' to the $\mathcal{M}\Sigma$ -expansion of Q_1 ;
2. Obtain a chase result $(Q_1')^{\mathcal{M}\Sigma}$ of Q_1' with $\Upsilon_{\mathcal{M}\Sigma}$;
3. If $((Q_1')^{\mathcal{M}\Sigma}$ is a trivial UCQ^\neq query
4. or $(Q_1')^{\mathcal{M}\Sigma} \sqsubseteq Q_2$) then output “yes”; else output “no.”

(Recall that [19] provides a containment test for the UCQ^\neq query $(Q_1')^{\mathcal{M}\Sigma}$ and CQ query Q_2 in line 4.)

We now show that the algorithm $\mathcal{M}\Sigma$ -CONTAINMENT DETERMINATION is correct for CQ weakly acyclic inputs. Our first observation is as follows.

PROPOSITION 5.2. *For a CQ weakly acyclic $\mathcal{M}\Sigma = (\mathbf{P}, \Sigma, \mathcal{V}, MV)$ and CQ query Q , let $(Q)^{\mathcal{M}\Sigma}$ be a chase result of the $\mathcal{M}\Sigma$ -expansion Q' of Q with $\mathcal{M}\Sigma$. Then:*

1. $(Q)^{\mathcal{M}\Sigma} \sqsubseteq Q$, and
2. $Q \sqsubseteq_{\mathcal{M}\Sigma} (Q)^{\mathcal{M}\Sigma}$. □

The proof of item 2 of Proposition 5.2 is by induction on the chase steps for Q' and $\Upsilon_{\mathcal{M}\Sigma}$, once we fix an instance I such that $\mathcal{V} \Rightarrow_{I, \Sigma} MV$. Specifically, the property in item 2 is an invariant for the output of each chase step of Q' with $\Upsilon_{\mathcal{M}\Sigma}$, for any fixed such I .

Our next observation concerns valuations for the query $(Q)^{\mathcal{M}\Sigma}$ and for *arbitrary* instances of schema \mathbf{P} . (The proof is by construction of each UCQ^\neq query $(Q)^{\mathcal{M}\Sigma}$.)

PROPOSITION 5.3. *Given a CQ weakly acyclic setting $\mathcal{M}\Sigma = (\mathbf{P}, \Sigma, \mathcal{V}, MV)$ and a CQ query Q . For any nontrivial chase result $(Q)^{\mathcal{M}\Sigma}$ of the $\mathcal{M}\Sigma$ -expansion of Q with $\mathcal{M}\Sigma$, denote by B^* all the relational atoms in the body of $(Q)^{\mathcal{M}\Sigma}$. Then for every instance I of schema \mathbf{P} and for each valuation ν for $(Q)^{\mathcal{M}\Sigma}$ and I , $\nu(B^*)$ is a Σ -valid base instance for \mathcal{V} and MV . □*

By Propositions 5.2–5.3, for CQ weakly acyclic instances $(\mathcal{M}\Sigma, Q_1, Q_2)$, all the chase results $(Q_1')^{\mathcal{M}\Sigma}$ are trivial UCQ^\neq queries (i.e., each of them is the empty set) iff (*) the answer to the input query Q_1 is empty on all Σ -valid base instances. Further, a $(Q_1')^{\mathcal{M}\Sigma} = \emptyset$ only if (*) holds. This justifies the “yes” output when the condition of line 3 of the algorithm evaluates to true.

Propositions 5.1 through 5.3 permit us to establish correctness of the algorithm $\mathcal{M}\Sigma$ -CONTAINMENT DETERMINATION for CQ weakly acyclic inputs:

THEOREM 5.1. *Given a CQ weakly acyclic instance $(\mathcal{M}\Sigma, Q_1, Q_2)$. Then $Q_1 \sqsubseteq_{\mathcal{M}\Sigma} Q_2$ if and only if for any one chase result $(Q_1')^{\mathcal{M}\Sigma}$ of the $\mathcal{M}\Sigma$ -expansion of Q_1 with $\mathcal{M}\Sigma$, either $(Q_1')^{\mathcal{M}\Sigma} = \emptyset$ or $(Q_1')^{\mathcal{M}\Sigma} \sqsubseteq Q_2$. □*

By this result, our solution to the certain-answer problem presented in Example 1.1, for the tuple (johnDoe, 50000), is correct for the setting of this example. (We solve that certain-answer problem via determining $\mathcal{M}\Sigma$ -conditional containment, as stipulated in Theorem 4.1.)

As discussed earlier, the approach of [31] is exactly the algorithm $\mathcal{M}\Sigma$ -CONTAINMENT DETERMINATION for the case $\Sigma = \emptyset$. The chase result $(Q_1')^{\mathcal{M}\Sigma}$, with $\Sigma = \emptyset$, is denoted in [31] by Q_1'' . We have shown that our extension of the approach of [31] to the cases where $\Sigma \neq \emptyset$ is not as simple as “just chasing Q_1'' with the input dependencies Σ .” In fact, even if we chase Q_1'' with our modified dependencies $\Sigma_{(\neq)}$, we are not guaranteed a correct output. (See Appendix F for all the details.) Thus, algorithm $\mathcal{M}\Sigma$ -CONTAINMENT DETERMINATION is not a trivial extension of the approach of [31].

We can also show that to chase the query Q_1' with the dependencies Σ , and to then chase the resulting query with the dependencies $\Phi_{(MV)}$, does not, in general, yield a correct determination of $Q_1 \sqsubseteq_{\mathcal{M}\Sigma} Q_2$ when we apply the unconditional-containment (\sqsubseteq) test. However, it is by construction of the dependencies $\Sigma_{(\neq)}$ that chasing Q_1' with $\Sigma_{(\neq)}$ (rather than Σ) only, followed by chase with the dependencies $\Phi_{(MV)}$ only, yields correct chase results for the purpose of determining $\mathcal{M}\Sigma$ -conditional containment for CQ weakly acyclic inputs.

Finally, we note that the presence of disequality atoms is critical to ensure correctness of our algorithm. Specifically, if disequality atoms are not introduced into either $\Sigma_{(\neq)}$ or the dependencies $\Phi_{(MV)}$, then the result of Proposition 5.3 no longer holds. (See Appendix G for all the details.) As a result, it is no longer clear how to ensure that the only-if direction of Theorem 5.1 (in case where $(Q_1')^{\mathcal{M}\Sigma} \neq \emptyset$) goes through.

6. FINDING ALL THE CERTAIN ANSWERS

The results of Sections 4–5 suggest an approach for finding all certain-answer tuples for CQ weakly acyclic inputs. For a k -ary query Q and a setting $\mathcal{M}\Sigma$, the approach is to generate all the k -ary tuples of values in $\text{consts}(\mathcal{M}\Sigma)$, and then for each such tuple, \bar{t} , to solve the certain-answer problem for Q, \bar{t} , and $\mathcal{M}\Sigma$, by using Theorem 4.1 and algorithm $\mathcal{M}\Sigma$ -CONTAINMENT DETERMINATION. By our results above, this approach is a correct algorithm for CQ weakly acyclic inputs. At the same time, its generate-and-test flavor may result in voluminous unnecessary computation for all those tuples \bar{t} that are *not* certain answers for the given input.

In this section we introduce an approach, called “view-verified data exchange,” which solves the same problem but is not based on the generate-and-test paradigm. As the name suggests, this approach is based on data exchange [13, 5, 4]. This approach is also intimately related to the techniques that we used in Section 5 to address $\mathcal{M}\Sigma$ -conditional query containment. Specifically, view-verified data exchange uses a modification of the dependencies $\Upsilon_{\mathcal{M}\Sigma}$ of Section 5, in which we do away with the disequality atoms in the dependencies. Due to the page limit, in this section we provide just a brief overview; all the details, including a full formalization and examples, can be found in Appendix J.

Given a CQ setting $\mathcal{M}\Sigma = (\mathbf{P}, \Sigma, \mathcal{V}, MV)$, the idea of view-verified data exchange is very natural: We borrow from the standard data-exchange framework, in that we

treat the relation symbols in \mathcal{V} as the “source schema” and the schema \mathbf{P} as the “target schema,” with “target constraints” Σ . Further, we treat natural tgds arising from the definitions of the views in \mathcal{V} as “source-to-target dependencies” Σ_{st} for this “data-exchange setting.” Then we could treat the set MV as a “source instance,” and pose the input query Q on the “target instances” that are determined by this data-exchange setting and by this source instance. (All the relevant formal definitions can be found in Appendices H–I.)

One special type of target instance used in data exchange is called “canonical universal solution” [13] for the given data-exchange setting and source instance. Such instances are obtained by chase of the source instance with the dependencies $\Sigma_{st} \cup \Sigma$, and can be used to represent, in the following precise sense, all target instances of interest. When Σ_{st} is a set of tgds, Σ is weakly acyclic, and Q is a UCQ query, the problem of computing certain answers for Q , w.r.t. the given data-exchange setting and source instance, can be solved via posing Q on a canonical universal solution [13]. It turns out that this result can be carried over directly to the problem of finding certain answers to a query in presence of a *materialized-view* setting, resulting in a sound and complete algorithm [27] under *OWA* for the CQ weakly acyclic cases of the problem.

Not surprisingly, the algorithm of [27] is not complete under *CWA*. (See Appendices H–I for the details.) In particular, applying the algorithm of [27] to our Example 1.1 would produce the empty set of certain-answer tuples. At the same time, using the results of Section 5 we can show that $\bar{t} = (\text{johnDoe}, 50000)$ is a certain answer for the setting of Example 1.1 under *CWA*. As it would be straightforward for attackers to obtain that tuple \bar{t} “from first principles,” our motivation was to come up with a correct algorithm for the *CWA* version of the problem of finding all certain query answers, as defined in Section 4.1. Our view-verified data exchange does qualify, by being a sound and complete algorithm for all CQ weakly acyclic instances under *CWA*.

We outline here the main idea of view-verified data exchange. (Due to the space limit, all the details can be found in Appendix J.) Just as in the approach of [27], we begin by obtaining a canonical universal solution, $J_{de}^{M\Sigma}$, for the data-exchange setting that arises naturally from the input instance $(M\Sigma, Q)$. We then apply to $J_{de}^{M\Sigma}$ *disjunctive chase*, as specified for our problem of $M\Sigma$ -conditional query containment, with two modifications. One, we chase the *instance* $J_{de}^{M\Sigma}$, essentially by treating it as the body of a CQ query. Two, we use in the chase a modification of the dependencies $\Upsilon_{M\Sigma} = \Phi_{(MV)} \cup \Sigma_{(\neq)}$ of Section 5. The idea of this modification of $\Upsilon_{M\Sigma}$ is that we do *not* normalize the left-hand side of any dependency. One consequence of this choice is that disequalities do not arise in the right-hand side of any resulting dependency. (In particular, Σ remains unmodified, rather than giving rise to $\Sigma_{(\neq)}$ as in Section 5.) We show that disequalities are not necessary for correctness of the approach to the problem of finding certain-query answers. Intuitively, the instances that we obtain in the chase are used to characterize only Σ -valid instances for \mathcal{V} and MV , rather than all possible instances of schema \mathbf{P} . (In the problem of Section 5, the chase enforces constraints that ensure that Proposition 5.3 holds for $(Q_1)^{M\Sigma}$ on *all* instances of schema \mathbf{P} .)

Finally, the view-verified data-exchange approach obtains a set of answers without nulls to the input query Q on *each* of the instances in the chase result; the output is then the intersection of these sets. We have shown that for all CQ weakly acyclic inputs, the output of this approach is well defined and is the set of all certain answers to Q w.r.t. the setting $M\Sigma$. That is:

THEOREM 6.1. *View-verified data exchange is a sound and complete algorithm for finding certain answers for all CQ weakly acyclic instances under CWA.* \square

Interestingly, in view-verified data exchange one cannot always find all the certain answers correctly if one does the chase “in stages.” That is, chase only with the input dependencies Σ , followed by chase only with the “ MV -induced dependencies,” does not always yield a correct solution. The reverse order of the “stages” is not guaranteed to work either. See Appendix L for the details.

7. COMPLEXITY OF THE PROBLEMS

In this section we consider the complexity of the CQ weakly acyclic cases of the three problems defined in Section 4.1. Our main focus is on the security-relevant complexity measure introduced in [31]. Due to the page limit, the exposition in this section is just an outline of the results; Appendices J and M provide the details.

Generally, in studying the complexity of the certain-query-answer problem of Definition 4.2, it is natural to build on the results of [1], which were established w.r.t. the complexity measures introduced in [28]. For instance, for the CQ weakly acyclic case of the problem of Definition 4.2, it is straightforward to obtain membership in coNP for the “data complexity” of the problem, that is, for the assumption that the set MV is the only non-fixed part of $(M\Sigma, Q, \bar{t})$. Then one can use the coNP-hardness result of [1] for the special case $\Sigma = \emptyset$, to arrive at the overall coNP completeness of the CQ weakly acyclic case of the problem of Definition 4.2 w.r.t. the data-complexity measure of [28].

Given the security focus of this current work, we concentrate here on a complexity measure that extends naturally that of [31]. Zhang and Mendelzon in [31] assumed for their “conditional-containment” problem that the base schema and the view definitions are fixed, whereas the set of view answers MV and the queries posed on the base schema in presence of MV can vary. (This assumption is natural in, e.g., database-access control [7], where access-control views are typically defined once for each (class of) users, and where the only frequently changing parts of the problem instance would be the view answers, MV , seen by the users, as well as the “secret queries” Q .) [31] did not consider dependencies on the base schema; we follow the standard data-exchange assumption, see, e.g., [13], that the given dependencies are fixed, rather than being part of the problem input.

Under this complexity metric, we consider first the complexity of the certain-query-answer problem (Definition 4.2) and of the $M\Sigma$ -conditional containment problem (Definition 4.3). Given the tight relationship between these problems (see Theorem 4.1), specifically between their CQ weakly acyclic cases, we can view the two problems together, using the following “grid”:

1. The CQ weakly acyclic case of the certain-query-answer problem with $\Sigma = \emptyset$;
2. The general (i.e., $\Sigma \neq \emptyset$ is possible) CQ weakly acyclic case of the certain-query-answer problem;

3. The CQ weakly acyclic case of the $\mathcal{M}\Sigma$ -conditional-containment problem with $\Sigma = \emptyset$; and
4. The general (i.e., $\Sigma \neq \emptyset$ is possible) CQ weakly acyclic case of $\mathcal{M}\Sigma$ -conditional containment.

With the help of Theorem 4.1, it is easy to show that Problem 1 above is a special case of each of Problems 2 and 3, and that each of the latter problems is, in turn, a special case of Problem 4.

Using these relationships, we have shown that each of Problems 1–4 is Π_2^P complete w.r.t. our extension, above, of the complexity measure of [31]. These four results are immediate from the results of Theorems 7.1–7.2, to follow, and from our observations above on the inclusions between the four problems.

THEOREM 7.1. *The certain-query-answer problem of Definition 4.2 is Π_2^P hard for CQ input instances $(\mathcal{M}\Sigma, Q, \bar{t})$ in which $\Sigma = \emptyset$ in the setting $\mathcal{M}\Sigma$, under the assumption that everything in the instance $(\mathcal{M}\Sigma, Q, \bar{t})$ is fixed except for Q , \bar{t} , and the set MV in $\mathcal{M}\Sigma$. \square*

(It is easy to show that in the setting of Theorem 7.1, it is enough to consider problem instances in which the size of the tuple \bar{t} in $(\mathcal{M}\Sigma, Q, \bar{t})$ is the arity of the query Q . See Appendix M for the details.)

The result of Theorem 7.1 is by reduction from the $\forall\exists$ -CNF problem, which is known to be Π_2^P complete [26]. (Please see Appendix M for a detailed proof.) We start off from the reduction that was used by Millstein and colleagues in [21] for the problem of query containment for data-integration systems. We modify the reduction of [21] in the spirit that is similar to the modification of that reduction (of [21]) as suggested in [31]. (Recall that the full version of [31], including any of its proofs, has never been published.) The goal of our modification is to comply with our assumptions about the input size, specifically with the assumption that the input view definitions are fixed. (In [21] it is assumed that both the queries and the view definitions can vary.)

THEOREM 7.2. *The $\mathcal{M}\Sigma$ -conditional containment problem of Definition 4.3 is in Π_2^P for CQ weakly acyclic input instances $(\mathcal{M}\Sigma, Q_1, Q_2)$, under the assumption that everything in the instance $(\mathcal{M}\Sigma, Q_1, Q_2)$ is fixed except Q_1 , Q_2 , and the set MV in $\mathcal{M}\Sigma$. \square*

(The proof is straightforward from the results of Section 5, specifically of Proposition 5.1.)

Finally, consider the complexity of the CQ weakly acyclic case of the problem of finding all certain-answer tuples. Observe first that, in the special case where Q is a Boolean query, the problem of finding all certain-answer tuples reduces to the certain-query-answer problem for the same $\mathcal{M}\Sigma$ and Q , with $\bar{t} = ()$. Now recall that the view-verified data exchange of Section 6 is a sound and complete algorithm for the (general) CQ weakly acyclic case of this problem. Using this algorithm, we establish a singly-exponential upper bound on the time complexity of the problem, under the same complexity measure as above, that is, assuming that in each instance $(\mathcal{M}\Sigma, Q)$, everything is fixed except for the query Q and for the set MV in $\mathcal{M}\Sigma$. Further, under the same complexity measure, solving the CQ weakly acyclic case of the problem is in PSPACE (provided the algorithm does certain things on-the-fly). Note that the output size is up to exponential in the arity of the input query Q . See Appendix J for all the details.

8. REFERENCES

- [1] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *PODS*, 1998.
- [2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- [3] A. V. Aho, Y. Sagiv, and J. D. Ullman. Equivalences among relational expressions. *SIAM J. Comput.*, 8:218–246, 1979.
- [4] M. Arenas, P. Barceló, L. Libkin, and F. Murlak. *Relational and XML Data Exchange*. Morgan & Claypool, 2010.
- [5] P. Barceló. Logical foundations of relational data exchange. *SIGMOD Record*, 38(1):49–58, 2009.
- [6] C. Beeri and M. Y. Vardi. The implication problem for data dependencies. In *ICALP*, pages 73–85, 1981.
- [7] E. Bertino, G. Ghinita, and A. Kamra. Access control for databases: Concepts and systems. *Foundations and Trends in Databases*, 3(1-2):1–148, 2011.
- [8] A. Brodsky, C. Farkas, and S. Jajodia. Secure databases: Constraints, inference channels, and monitoring disclosures. *IEEE TKDE*, 12(6):900–919, 2000.
- [9] A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, 1977.
- [10] A. Deutsch. *XML Query Reformulation over Mixed and Redundant Storage*. PhD thesis, Univ. Pennsylvania, 2002.
- [11] A. Deutsch, A. Nash, and J. Rimmel. The chase revisited. In *PODS*, pages 149–158, 2008.
- [12] A. Deutsch and V. Tannen. Optimization properties for classes of conjunctive regular path queries. In *DBPL*, 2001.
- [13] R. Fagin, P. Kolaitis, R. Miller, and L. Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336:89–124, 2005.
- [14] W. Fan and F. Geerts. *Foundations of Data Quality Management*. Morgan & Claypool, 2012.
- [15] A. Fuxman, P. G. Kolaitis, R. J. Miller, and W.-C. Tan. Peer data exchange. *ACM TODS*, 31(4):1454–1498, 2006.
- [16] S. Greco, C. Molinaro, and F. Spezzano. *Incomplete Data and Data Dependencies in Relational Databases*. Morgan & Claypool, 2012.
- [17] T. Imielinski and W. Lipski. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.
- [18] D. S. Johnson and A. C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comput. Syst. Sci.*, 28(1):167–189, 1984.
- [19] A. Levy, A. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *PODS*, 1995.
- [20] G. Miklau and D. Suciu. A formal analysis of information disclosure in data exchange. *JCSS*, 73(3):507–534, 2007.
- [21] T. D. Millstein, A. Y. Halevy, and M. Friedman. Query containment for data integration systems. *JCSS*, 66, 2003.
- [22] R. Motwani, S. U. Nabar, and D. Thomas. Auditing SQL queries. In *ICDE*, pages 287–296, 2008.
- [23] S. U. Nabar, K. Kenthapadi, N. Mishra, and R. Motwani. A survey of query auditing techniques for data privacy. In *Privacy-Preserving Data Mining*, pages 415–431, 2008.
- [24] S. Rizvi, A. O. Mendelzon, S. Sudarshan, and P. Roy. Extending query rewriting techniques for fine-grained access control. In *SIGMOD Conference*, pages 551–562, 2004.
- [25] C. E. Shannon. Communication theory of secrecy systems. *Bell Syst. Techn. J.*, 28:656–715, 1949.
- [26] L. J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.
- [27] K. Stoffel and T. Studer. Provable data privacy. In *DEXA*, pages 324–332, 2005.
- [28] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *STOC*, pages 137–146, 1982.
- [29] R. W. Yip and K. N. Levitt. Data level inference detection in database systems. In *CSFW*, pages 179–189, 1998.
- [30] X. Zhang and M. Özsoyoglu. Implication and referential constraints: A new formal reasoning. *IEEE TKDE*, 9, 1997.
- [31] Z. Zhang and A. O. Mendelzon. Authorization views and conditional query containment. In *ICDT*, 2005.

APPENDIX

A. CERTAIN QUERY ANSWERS:

EXAMPLE WITH $\Sigma = \emptyset$

In this appendix we show an example with $\Sigma = \emptyset$, of an input instance for the certain-query-answer problem of Definition 4.2 and for the problem of finding the set of certain query answers w.r.t. a materialized-view setting, see Section 4.1. This example is to be used as an illustration in later appendices, e.g., in Appendix B.

EXAMPLE A.1. Consider a relation E (for **Employee**), which is used for storing information about employees of a company. Let the attributes of E be **Name**, **Dept** (for the departments in which the employees work), and **Salary**: $E(\text{Name}, \text{Dept}, \text{Salary})$.

We assume that no integrity constraints hold on the database schema \mathbf{P} containing the relation E . (In particular, the only primary key of E is all its attributes.) Thus, the set Σ of dependencies holding on the schema \mathbf{P} is the empty set.

Let a query Q ask for the salaries of all the employees. We can formulate the query Q in SQL as

(Q): SELECT DISTINCT Name, Salary FROM E;

The query Q is a CQ query, which can be expressed in Datalog as follows:

$Q(X, Z) \leftarrow E(X, Y, Z)$.

Consider two views, V and W , that are defined for some class(es) of users on the schema \mathbf{P} . The view V returns the departments for each employee, and the view W returns the salaries in each department. The Datalog definitions of these CQ views are as follows. (Please see Example 1.1 for the SQL definitions of V and W .)

$V(X, Y) \leftarrow E(X, Y, Z)$.
 $W(Y, Z) \leftarrow E(X, Y, Z)$.

Suppose that some user(s) are authorized to see the answers to V and W , and that at some point in time the user(s) can see the following set MV of answers to these views.

$MV = \{ V(\text{johnDoe}, \text{sales}), W(\text{sales}, 50000) \}$.

Then one “conjunctive fact-expression” \mathcal{C}_{MV} (see Section 4.3) that the user(s) can put together based on this instance MV is

$\mathcal{C}_{MV} = V(\text{johnDoe}, \text{sales}) \text{ AND } W(\text{sales}, 50000)$.

Let $\bar{t} = (\text{johnDoe}, 50000)$ be the tuple that the user hypothesizes is in the answer to the query Q on all the instances of the relation **Emp** that satisfy the (empty set of) dependencies Σ and that generate the above instance MV . Observe that the tuple \bar{t} is made up from values **johnDoe** and **50000**, which “are generated by” the expression \mathcal{C}_{MV} . Thus, knowing the associations between the values in the tuple \bar{t} and the respective attribute names in MV , we can “put together” this expression \mathcal{C}_{MV} and this tuple \bar{t} as a SQL query, **Rvw**, in terms of the views V and W and in presence of the constants from the instance MV , as follows:

(Rvw): SELECT DISTINCT Name, Salary FROM V, W
 WHERE Name = 'johnDoe' AND V.Dept = W.Dept
 AND V.Dept = 'sales' AND Salary = '50000';

That is, by defining the query **Rvw** we formalize the rather natural process of the user “putting together” tuples in the available instance MV and of his then using some of the values from the selected tuples to put forth a tuple of constants that is hypothetically in the answer to the query Q . We note that **Rvw** is defined by a CQ query:

$R_{vw}(\text{johnDoe}, 50000) \leftarrow V(\text{johnDoe}, \text{sales}),$
 $W(\text{sales}, 50000)$.

By definition of **Rvw**, the above tuple $\bar{t} = (\text{johnDoe}, 50000)$ is the only possible answer to **Rvw** on all possible instances of the relations V and W . It is easy to see that this answer to the query **Rvw** is compatible with (i.e., can be obtained by asking the query **Rvw** on) the above instance MV . (Intuitively, this is true because we have constructed **Rvw** from the tuples in the above instance MV .) \square

B. RELATIONSHIP BETWEEN THE CERTAIN-QUERY-ANSWER PROBLEM W.R.T. A SETTING AND THE QUERY-CONTAINMENT PROBLEM W.R.T. A SETTING

In this appendix we provide the technical details on the main result of Section 4.3. That result, Theorem 4.1, establishes a direct relationship between the certain-query-answer problem for a given Q , \bar{t} , and a valid CQ setting $\mathcal{M}\Sigma$, and the problem of $\mathcal{M}\Sigma$ -conditional containment for Q' and Q , for the same Q and $\mathcal{M}\Sigma$. Here, the query Q' is constructed from the given Q , \bar{t} , and $\mathcal{M}\Sigma$. The proof of Theorem 4.1 is immediate from Theorem B.2, see Section B.3 of this appendix.

A relationship similar to that of Theorem 4.1 was observed in [1] for the dependency-free case under OWA. In contrast, our result holds under CWA and in presence of dependencies on the schema \mathbf{P} in the setting $\mathcal{M}\Sigma$.

B.1 The Intuition

The intuition for the relationship between the two problems can be illustrated via Example A.1. That is, we formalize the thought process of the presumed attackers concerning the answers to “secret queries” [20] (such as the query Q in Example A.1) that are posed on a proprietary database. The attackers know a materialized-view setting $\mathcal{M}\Sigma = (\mathbf{P}, \Sigma, \mathcal{V}, MV)$ and the definition of a query Q , and come up with candidate certain answers to Q in this setting $\mathcal{M}\Sigma$. (This is, informally, the idea of the problem of database access control, see, e.g., [7].)

We argue that the approach of putting together the tuples in the view answers is natural for the presumed attackers to use. Indeed, in any specific instance of the problem of the certain query answer w.r.t. a materialized-view setting, attackers deal directly with a ground instance MV . They know that MV is a set of answers to the “access-policy” views \mathcal{V} on the underlying instance of interest. Thus, intuitively, a question that is natural for the attackers to ask is which values in $\text{consts}(\mathcal{M}\Sigma)$ can be put together to form an answer to the secret

query Q , on all possible underlying instances of interest. (In general, the attackers could consider in their pursuit not just values in $\text{consts}(\mathcal{M}\Sigma)$, but also constants mentioned in the queries for \mathcal{V} and in the secret query Q . It is straightforward to reflect this in our setting, by adding extra head arguments to the definitions of the respective views. Thus, we do not explicitly consider this extension in this paper.)

How can this certain-query-answer question be answered deterministically, as required by Definition 4.2? A natural approach would be to put together a query, call it R , in terms of the relations in the instance MV , and to then prove that R is “contained,” in some precise sense (in particular, w.r.t. the views in \mathcal{V}), in the secret query Q . We will be referring to all queries R over schema \mathcal{V} as “rewritings” (in terms of \mathcal{V}), as indeed they would be defined in terms of the relation symbols in \mathcal{V} . (Another reason to refer to such queries R as “rewritings” is that we will need to define their expansions shortly.) Hence our name for this approach to solving the problem of whether a ground tuple \bar{t} is a certain answer to a query Q w.r.t. a setting $\mathcal{M}\Sigma$. As we will see, this approach determines precisely containment between two queries w.r.t. the given setting $\mathcal{M}\Sigma$. Here, one of the two queries in question is the query Q provided in the problem input, and the other query is the “expansion” [19] of a rewriting R , with head \bar{t} , such that the definition of R is obtained from $\mathcal{M}\Sigma$.

A challenge arises immediately when attackers pursue this train of thought: In the set $R(MV)$ of answers to a rewriting R on an instance MV , not all the tuples in $R(MV)$ would necessarily be in the answer to the secret query Q . That is, the formal containment that we are looking for would not hold for all rewritings R . (As an illustration, suppose that in some instance of the certain-query-answer problem w.r.t. a setting, the input query Q returns names of employees with high salaries, and R returns names of employees in the accounting department. Clearly, the answer to R is not necessarily a subset of the answer to Q , on any particular database of interest.)

At the same time, for each individual tuple $\bar{t} \in R(MV)$, it makes sense to ask the question of whether the query $R(\bar{t})$ is contained in Q in the appropriate precise sense. The intuition is that $R(\bar{t})$ is the result of binding the head vector of R to a tuple, \bar{t} , in the relation $R(MV)$; as a result, \bar{t} is the only answer to $R(\bar{t})$ on the instance MV . We focus on such rewritings $R(\bar{t})$ in this approach.

B.2 Defining the Rewriting Approach

We now formalize the “rewriting approach” to determining whether a given ground tuple \bar{t} is a certain answer to a given query Q w.r.t. a given materialized-view setting $\mathcal{M}\Sigma$. As outlined in Section B.1, the intuition is that this rewriting approach works by determining containment between two queries w.r.t. the setting $\mathcal{M}\Sigma$, such that each of the two queries is obtained from some combination of the given inputs Q , \bar{t} , and $\mathcal{M}\Sigma$. Our intent is to tie the definitions of rewritings that attackers can formulate on view answers, to components of the given setting $\mathcal{M}\Sigma$. After defining rewritings of the form $R(\bar{t})$, we recall the standard notion of *expansion* of a view-based rewriting [19]; an expansion of a rewriting is its equivalent reformulation over the schema \mathbf{P} used to define the query Q . We then formalize the rewriting

approach, using the notion of containment of queries over the same schema w.r.t. a set of view answers MV and a set of dependencies Σ .

Head-instantiated rewriting $R(\bar{t})$. Intuitively, an attackers’ goal in this approach is to form candidate answers, \bar{t} , to the secret query Q , by using constants that are in $\text{consts}(\mathcal{M}\Sigma)$ and that thus presumably originate from the actual instance I of interest, $\mathcal{V} \Rightarrow_{I, \Sigma} MV$. (That is, the instance I is the actual proprietary database, of interest to the attackers, that has been used to generate the instance MV .) Observe that not all \mathcal{V} -based rewritings could be used toward this goal. Consider, for instance, a rewriting $R_f(f) \leftarrow V(X)$, defined using a constant f and a subgoal $V(X)$ for a view V and variable X . Clearly, regardless of the contents of the set MV of answers to the view V , the answer $R_f(f)(MV)$ to R_f on MV is always the set $\{ (f) \}$. To rule out rewritings such as $R_f(f)$, we define a desirable type of rewritings as follows.

For an integer $k \geq 0$ and for a k -tuple \bar{t} of constants, consider a safe k -ary CQ query R over schema \mathcal{V} and with head vector \bar{t} . We say that R is a *head-instantiated rewriting* for \bar{t} iff there exists a safe k -ary CQ query $R^{(g)}(\bar{X})$ over the schema \mathcal{V} , called a *grounding rewriting* for R , that satisfies two conditions. First, the head vector \bar{X} of $R^{(g)}$ does not include constants. Second, there exists a mapping, h , that maps all the elements of \bar{X} to constants and that maps the remaining terms in $R^{(g)}$ to themselves, such that the rewriting that results from applying h to the definition of $R^{(g)}$ is exactly R .

EXAMPLE B.1. Consider rewritings R_{vw} and \tilde{R}_{vw} that use constants c , d , and f . (\tilde{R}_{vw} also uses a variable Z .)

$$\begin{aligned} R_{vw}(c, f) &\leftarrow V(c, d), W(d, f). \\ \tilde{R}_{vw}(c, f) &\leftarrow V(c, Z), W(Z, f). \end{aligned}$$

Suppose that c , d , and f stand for ‘johnDoe’, ‘sales’, and ‘50000’, respectively; then R_{vw} is the rewriting R_{vw} of Example A.1. By applying this “translation of constants” to the instance MV of Example A.1, we obtain an instance $MV = \{V(c, d), W(d, f)\}$.

Each of R_{vw} and \tilde{R}_{vw} is a head-instantiated rewriting for (c, f) , as the respective grounding rewritings are

$$\begin{aligned} R_{vw}^{(g)}(X, Y) &\leftarrow V(X, d), W(d, Y). \\ \tilde{R}_{vw}^{(g)}(X, Y) &\leftarrow V(X, Z), W(Z, Y). \end{aligned} \quad \square$$

By definition, for each head-instantiated rewriting R for a tuple \bar{t} , the answer to R on an instance I of schema \mathcal{V} is nonempty (and is exactly $\{ \bar{t} \}$) iff there exists a valuation from the body of R onto a subset I' of I such that $\text{adom}(I')$ contains all the constants in \bar{t} . Further, consider an arbitrary safe CQ query R'' over schema \mathcal{V} , and consider any instance MV such that $R''(MV) \neq \emptyset$. Then for each tuple \bar{t} in $R''(MV)$, the result $R''(\bar{t})$ of binding the head vector of R'' to \bar{t} (while consistently renaming the terms in the body of R'' as well) is a head-instantiated rewriting for \bar{t} , such that the answer to $R''(\bar{t})$ on the instance MV is not empty (and is, obviously, exactly $\{ \bar{t} \}$).

Expansion of a rewriting. We now take a step back, from head-instantiated rewritings to general CQ rewritings, to recall the standard notion of expansion of

a CQ rewriting [19]. First, given a set of views \mathcal{V} and a ground instance I of schema \mathbf{P} , consider an instance over schema $\mathcal{V} \cup \mathbf{P}$, which results from adding to I the relation $V(I)$ for each relation symbol $V \in \mathcal{V}$. We call the latter instance *the \mathcal{V} -enhancement of I* , and denote it by $I^{(+\mathcal{V})}$. Now given a rewriting R over the schema \mathcal{V} , consider a query, R' , over the schema \mathbf{P} such that for each instance I of \mathbf{P} we have $R'(I) = R(I^{(+\mathcal{V})})$. We call such a query R' *an expansion of R (over \mathbf{P})*, and denote it by R^{exp} . We will use the following straightforward property of R^{exp} :

PROPOSITION B.1. *For a set \mathcal{V} of views over schema \mathbf{P} : Let R be a query over \mathcal{V} such that R^{exp} exists, and let MV be an instance of schema \mathcal{V} . Then for each instance I of schema \mathbf{P} such that $\mathcal{V} \Rightarrow_{I, \emptyset} MV$, we have $R^{exp}(I) = R(MV)$. \square*

In case where \mathcal{V} is a set of CQ views and R is a CQ query over \mathcal{V} , the standard process in the literature of constructing R^{exp} is [19] to replace each subgoal of R with the body of the query for the corresponding relation symbol in \mathcal{V} . In this process, care is taken to perform two operations on each query, of the form $V(\bar{X}) \leftarrow body_{(V)}$, whose body in R^{exp} corresponds to a subgoal of R of the form $V(\bar{Z})$. First, we *bind the arguments of the query for V to the vector \bar{Z}* , in two steps, (A) and (B). The step (A) is to extend the homomorphism,⁵ h , that maps each element of the head vector \bar{X} of the query for V to the same-position element of \bar{Z} , to a homomorphism $h_{V(\bar{Z})}$, whose domain is the set of all arguments of $body_{(V)}$, such that $h_{V(\bar{Z})}$ is the identity mapping for each value that is not in the domain of h . Then, (B) is to apply $h_{V(\bar{Z})}$ to $body_{(V)}$, with conjunction of relational atoms $h_{V(\bar{Z})}(body_{(V)})$ as the output. Second, before conjoining $h_{V(\bar{Z})}(body_{(V)})$ with the current body, $body_{R^{exp}}$, of the query R^{exp} , we rename all the variables in $h_{V(\bar{Z})}(body_{(V)})$ consistently into “fresh” variables not occurring in $body_{R^{exp}}$. The query R^{exp} that is obtained by this two-step process is (i) an expansion of R over \mathbf{P} , and is (ii) unique up to variable renaming.

Conditional containment: We can now use containment to directly relate a rewriting R , via R^{exp} , to the given query Q . The notion of containment we will use is that of Definition 4.3 in Section 4.1.

For notational convenience in the results to follow, we now introduce Σ -conditional containment of a rewriting in a query modulo a set of views and a set of answers to the views: For a rewriting R over \mathcal{V} such that R^{exp} exists, and for a query Q over \mathbf{P} , we say that R is Σ -conditionally contained in Q w.r.t. MV and modulo \mathcal{V} , denoted $R \sqsubseteq_{\Sigma, MV, \mathcal{V}} Q$, iff $R^{exp} \sqsubseteq_{\mathcal{M}\Sigma} Q$ holds.

The rewriting approach: We are now ready to specify the rewriting approach to the problem of determining whether a tuple \bar{t} is a certain answer to a query Q w.r.t. a materialized-view setting $\mathcal{M}\Sigma$. For an instance MV of schema \mathcal{V} , we say that a head-instantiated rewriting $R(\bar{t})$ is *MV -validated* iff the set $R(\bar{t})(MV)$ is not the empty set. (The rewritings R_{vw} and \tilde{R}_{vw} of

Example B.1 are both MV -validated.) Given a valid⁶ materialized-view setting $\mathcal{M}\Sigma$ with set of view answers MV , a k -ary ($k \geq 0$) query Q , and a k -ary tuple \bar{t} of constants in $consts(\mathcal{M}\Sigma)$, the *rewriting approach to the certain-query-answer problem for Q and \bar{t} in $\mathcal{M}\Sigma$* is to find an MV -validated head-instantiated rewriting R for \bar{t} such that $R \sqsubseteq_{\Sigma, MV, \mathcal{V}} Q$. This approach is sound:

PROPOSITION B.2. *Given a valid materialized-view setting $\mathcal{M}\Sigma = (\mathbf{P}, \Sigma, \mathcal{V}, MV)$ and a query Q of arity $k \geq 0$. Let \bar{t} be a k -tuple of values from $consts(\mathcal{M}\Sigma)$. Suppose that there exists an MV -validated head-instantiated rewriting R for \bar{t} such that $R \sqsubseteq_{\Sigma, MV, \mathcal{V}} Q$. Then \bar{t} is a certain answer to Q w.r.t. $\mathcal{M}\Sigma$. \square*

The proof is very simple: Any rewriting R satisfying the conditions of Proposition B.2 must have \bar{t} as its only answer on the given instance MV . Thus, by Proposition B.1, R^{exp} (which exists because the containment $R \sqsubseteq_{\Sigma, MV, \mathcal{V}} Q$ is stated in Proposition B.2 to be well defined) has \bar{t} as its only answer on all instances I such that $\mathcal{V} \Rightarrow_{I, \Sigma} MV$. From the containment $R^{exp} \sqsubseteq_{\mathcal{M}\Sigma} Q$ we conclude that on all such instances I , the tuple \bar{t} is an element of the set $Q(I)$. The claim of Proposition B.2 follows.

B.3 One Rewriting Is Enough

Suppose that we are given a materialized-view setting $\mathcal{M}\Sigma$ and a k -ary ($k \geq 0$) query Q . One (e.g., attackers) can generate all k -tuples \bar{t} with values in $consts(\mathcal{M}\Sigma)$. Then, Proposition B.2 gives the attackers a tool for testing each such \bar{t} as a certain-answer tuple to Q w.r.t. $\mathcal{M}\Sigma$, assuming that the attackers can come up with an “appropriate” rewriting R for each \bar{t} , and that there exists an algorithm for checking the containment $R \sqsubseteq_{\Sigma, MV, \mathcal{V}} Q$ for each such R and \bar{t} . We will consider in the next subsection some such algorithms. However, in this current subsection we show that to solve this generate-and-test problem for a given instance $\mathcal{M}\Sigma$, it is not necessary to also generate various bodies for rewritings R . Each valid $\mathcal{M}\Sigma$ is associated with a single CQ rewriting for each \bar{t} , with all these rewritings (for $\mathcal{M}\Sigma$) having the same body. The main result of this subsection is that for all CQ instances $\mathcal{M}\Sigma$, these rewritings alone can be used to capture *exactly* the set of all certain answers to the input query.

Intuitively, we are to construct the desired rewritings from the facts in the instance MV given as part of $\mathcal{M}\Sigma$. Indeed, by the requirement that MV in each $\mathcal{M}\Sigma$ be a ground instance, each fact in MV can be viewed equivalently as a relational atom whose all arguments are constants. Given a fixed MV and a k -ary ($k \geq 0$) tuple \bar{t} of values from $consts(\mathcal{M}\Sigma)$, we say that a CQ rewriting R with head vector \bar{t} is an *MV -induced rewriting for \bar{t}* iff each subgoal of R is a fact in MV . Further, an MV -induced rewriting R for \bar{t} is a *maximal MV -induced rewriting for \bar{t}* iff each fact in MV is also a subgoal of R . In Example B.1, R_{vw} is a maximal MV -induced rewriting for the tuple (c, f) , and \tilde{R}_{vw} is not an MV -induced rewriting.

We now list useful properties of MV -induced rewritings.

⁶The view-verified data-exchange approach of Appendix J can be used as a sound and complete algorithm for determining whether a given CQ weakly acyclic materialized-view setting is valid.

⁵It is easy to show that if such a h cannot be constructed, then R is unsatisfiable on all instances of the schema \mathcal{V} .

PROPOSITION B.3. *Given a valid materialized-view setting $\mathcal{M}\Sigma = (\mathbf{P}, \Sigma, \mathcal{V}, MV)$. For a $k \geq 0$, let \bar{t} , \bar{t}_1 , and \bar{t}_2 be k -tuples of values from $\text{consts}(\mathcal{M}\Sigma)$, for the MV in $\mathcal{M}\Sigma$. Then:*

- (1) *Each MV -induced rewriting R for \bar{t} is an MV -validated head-instantiated rewriting for \bar{t} whenever each element of \bar{t} occurs in the body of R ;*
- (2) *For each \bar{t} , there is exactly one maximal MV -induced rewriting, which is an MV -validated head-instantiated rewriting for \bar{t} ; and*
- (3) *The maximal MV -induced rewritings for \bar{t}_1 and for \bar{t}_2 have the same body, for all choices of \bar{t}_1 and \bar{t}_2 .*

□

Note 1. In case where some constants in $\text{consts}(\mathcal{M}\Sigma)$ are in definitions of the views in \mathcal{V} but are not in MV , all the claims of Proposition B.3 still go through once we modify the view definitions by adding all their body constants into their head vectors. This fix for this case also works for all the other results of this appendix that deal with head-instantiated rewritings for tuples \bar{t} constructed from the elements of the set $\text{consts}(\mathcal{M}\Sigma)$.

The next result says that when we have the maximal MV -induced rewriting for some tuple \bar{t} of values from $\text{consts}(\mathcal{M}\Sigma)$, then we do not need to consider any other head-instantiated rewritings for \bar{t} in our rewriting approach. (The proof is straightforward and is omitted.)

PROPOSITION B.4. *Given a valid CQ materialized-view setting $\mathcal{M}\Sigma = (\mathbf{P}, \Sigma, \mathcal{V}, MV)$ and a query Q of arity $k \geq 0$. Let \bar{t} be a k -tuple of values from $\text{consts}(\mathcal{M}\Sigma)$. Let R be an MV -validated head-instantiated rewriting for \bar{t} such that $R \sqsubseteq_{\Sigma, MV, \mathcal{V}} Q$. Then for the maximal MV -induced rewriting $R_{\bar{t}}^*$ for \bar{t} , we have $R_{\bar{t}}^* \sqsubseteq_{\Sigma, MV, \mathcal{V}} Q$.*

□

The following result says that maximal MV -induced rewritings alone can be used to capture exactly the certain answers to queries w.r.t. CQ materialized-view settings. This result is an immediate corollary of Propositions B.3 and B.4.

THEOREM B.1. *Given a valid CQ materialized-view setting $\mathcal{M}\Sigma$ and a query Q of arity $k \geq 0$. For a k -tuple \bar{t} of values from $\text{consts}(\mathcal{M}\Sigma)$: The tuple \bar{t} is a certain answer to Q w.r.t. $\mathcal{M}\Sigma$ iff for the maximal MV -induced rewriting $R_{\bar{t}}^*$ for \bar{t} , we have $R_{\bar{t}}^* \sqsubseteq_{\Sigma, MV, \mathcal{V}} Q$.*

□

PROOF. *If:* The proof of this direction parallels the proof of Proposition B.2.

Only-If: By Definition 4.2, for the given tuple \bar{t} we have that \bar{t} is in the set $Q(I)$ for all instances I of schema \mathbf{P} such that $\mathcal{V} \Rightarrow_{I, \Sigma} MV$. By Proposition B.3, we have that \bar{t} is the only answer on the instance MV to the maximal MV -induced rewriting $R_{\bar{t}}^*$ for \bar{t} . Thus, for the expansion of $R_{\bar{t}}^*$, denote this expansion by $(R_{\bar{t}}^*)^{exp}$, we have by Proposition B.1 that for each instance I of schema \mathbf{P} such that $\mathcal{V} \Rightarrow_{I, \Sigma} MV$, we have $(R_{\bar{t}}^*)^{exp}(I) = \{\bar{t}\}$. (In more detail, we have by Proposition B.1 that for each instance J of schema \mathbf{P} such that $\mathcal{V} \Rightarrow_{J, \emptyset} MV$, we have $(R_{\bar{t}}^*)^{exp}(J) = \{\bar{t}\}$. The conclusion that $(R_{\bar{t}}^*)^{exp}(I) = \{\bar{t}\}$ for each instance I of schema \mathbf{P} such that $\mathcal{V} \Rightarrow_{I, \Sigma} MV$ follows from the fact that the set of

all such instances I is a subset of the set of all such instances J .) Thus, by the definitions of expansions of rewriting and of the containment $\sqsubseteq_{\Sigma, MV, \mathcal{V}}$, we obtain immediately that $R_{\bar{t}}^* \sqsubseteq_{\Sigma, MV, \mathcal{V}} Q$. □

It follows from Theorem B.1 that the converse of Proposition B.2 also holds. Hence we obtain the following result.

THEOREM B.2. *Given a valid CQ materialized-view setting $\mathcal{M}\Sigma$ and a query Q of arity $k \geq 0$. For a k -tuple \bar{t} of values from $\text{consts}(\mathcal{M}\Sigma)$: There exists an MV -validated head-instantiated rewriting R for \bar{t} such that $R \sqsubseteq_{\Sigma, MV, \mathcal{V}} Q$ iff \bar{t} is a certain answer to Q w.r.t. $\mathcal{M}\Sigma$.*

□

Note 2. In the light of Theorems B.1 and B.2, we can use the results of this current paper on $\mathcal{M}\Sigma$ -conditional query containment to determine correctly if a given ground k -tuple \bar{t} is a certain answer to the (k -ary) query Q w.r.t. the setting $\mathcal{M}\Sigma$, for the class of all problem instances where Q is a CQ query, and the materialized-view settings $\mathcal{M}\Sigma$ is valid CQ weakly acyclic. Moreover, we can also find all the certain answers to Q w.r.t. $\mathcal{M}\Sigma$ for the same class of instances (i.e., CQ queries and valid CQ weakly acyclic materialized-view settings), by first generating all the ground k -tuples of values in $\text{consts}(\mathcal{M}\Sigma)$, and by then determining for each such tuple whether it is a certain answer to Q w.r.t. $\mathcal{M}\Sigma$. By the results of this paper, the latter algorithm is sound and complete for this class of input instances under CWA.

C. CONDITIONAL CONTAINMENT FOR CQ QUERIES

Zhang and Mendelzon in [31] addressed the problem of letting users access authorized data, via rewriting the users' queries in terms of their authorization views. Toward that goal, [31] explored the notion of "conditional query containment." The results of [31] include a powerful reduction of the problem of testing conditional containment of CQ queries to that of testing *unconditional* containment of modifications of the queries. In this appendix we review these results of [31]. Appendix D provides an illustrative example of conditional query containment.

We begin by reviewing the definition of conditional containment of queries [31]. Some of the definitions here are restricted versions of the definitions given in Section 4. We provide the restricted definitions here for this appendix to be self contained.

Suppose that we are given a schema \mathbf{P} and a set \mathcal{V} of relation symbols not in \mathbf{P} , with each symbol (*view name*) $V \in \mathcal{V}$ of some arity $k_V \geq 0$. Each symbol $V \in \mathcal{V}$ is defined via a k_V -ary query on the schema \mathbf{P} . We call \mathcal{V} a *set of views on \mathbf{P}* , and call the query for each $V \in \mathcal{V}$ the *definition of the view V* , or the *query for V* . We assume that the query for each $V \in \mathcal{V}$ is associated with $(V \text{ in})$ the set \mathcal{V} . Consider a ground instance MV of schema \mathcal{V} ; we call MV a *set of view answers for \mathcal{V}* . Then for a ground instance I of schema \mathbf{P} , we say that I is a *valid instance (of \mathbf{P}) for \mathcal{V} and MV* [31] whenever for each $V \in \mathcal{V}$, the answer $V(I)$ to the query for V on the instance I is identical to the relation $MV[V]$ for V in the instance MV . For a given set MV of view answers for a set of views \mathcal{V} , we say that MV is a *valid*

set of view answers for \mathcal{V} whenever there exists at least one valid instance for \mathcal{V} and MV .

Now given queries Q_1 and Q_2 on the schema \mathbf{P} , we say that Q_1 is *conditionally contained in Q_2 w.r.t. $(\mathcal{V}$ and) MV* [31], denoted⁷ $Q_1 \sqsubseteq_{MV} Q_2$, if and only if the relation $Q_1(I)$ is a subset of the relation $Q_2(I)$ for each valid instance I for \mathcal{V} and MV .

It is easy to see that for all instances MV of all schemas \mathcal{V} , the containment $Q_1 \sqsubseteq Q_2$ is a sufficient condition for the containment $Q_1 \sqsubseteq_{MV} Q_2$. Not surprisingly, $Q_1 \sqsubseteq_{MV} Q_2$ does not imply $Q_1 \sqsubseteq Q_2$; something more sophisticated is clearly called for. The authors of [31] report the following powerful test for conditional containment of CQ queries. (We say that \mathcal{V} is a set of CQ views if the query for each $V \in \mathcal{V}$ is a CQ query.)

THEOREM C.1. [31] *Given a schema \mathbf{P} , a set of CQ views \mathcal{V} on \mathbf{P} , a valid set MV of view answers for \mathcal{V} , and CQ queries Q_1 and Q_2 on the schema \mathbf{P} . Then $Q_1 \sqsubseteq_{MV} Q_2$ if and only if for the UCQ^\neq query Q_1'' constructed for Q_1 by an algorithm given in [31], we have $Q_1'' \sqsubseteq Q_2$. \square*

Theorem C.1 reduces the problem of testing conditional containment of CQ queries, $Q_1 \sqsubseteq_{MV} Q_2$, to the problem of testing (unconditional) containment in Q_2 of a UCQ^\neq modification of Q_1 . The latter containment can be decided by a test due to [19]. The required modification of Q_1 is done by an intricate algorithm given in [31]. We outline here briefly the intuition for the construction of Q_1'' from Q_1 .

We say that an instance I of schema \mathbf{P} *underproduces* MV if, for at least one $V \in \mathcal{V}$, the relation $V(I)$ is a proper subset of the relation $MV[V]$. By definition, each valid instance for \mathcal{V} and MV does not underproduce MV .

The construction of Q_1'' from Q_1 proceeds in two steps. The first step guarantees that its output, a CQ query Q_1^* , has the empty answer on all instances of \mathbf{P} that underproduce MV . This goal is achieved by defining Q_1^* as having the same head vector as in Q_1 , and by (Q_1^*) having the body that is a conjunction of the body of Q_1 with the conjunction C_{MV}^{exp} defined in Section 4.3.

The output of the second step in the construction, a UCQ^\neq query Q_1'' , has the same property as Q_1^* does. In addition, for each instance, I , of schema \mathbf{P} such that I does *not* underproduce MV , and for each valuation, ν , from the query Q_1'' to I , all the facts in $\nu(\text{body}(Q_1''))$ collectively constitute a valid instance for \mathcal{V} and MV . (This is done by adding to the body of Q_1'' disjunctions, equalities, and/or disequalities based on homomorphisms from the normalized bodies of the views in \mathcal{V} to the body of (the current version of) Q_1'' . The body of a CQ query is *normalized* whenever its relational part has only one occurrence of each variable and of each constant, and all the equalities between variables and/or constants are enforced by explicit equality atoms.)

The result of Theorem C.1 is shown in [31] to follow from these properties of Q_1'' and from the fact that for all valid instances I for \mathcal{V} and MV , $Q_1''(I) = Q_1(I)$.

⁷To avoid overcrowding the symbol \sqsubseteq , we assume that in the notation \sqsubseteq_{MV} , the name MV of an instance of schema \mathcal{V} uniquely identifies the relevant set \mathcal{V} .

D. EXAMPLE OF QUERY CONTAINMENT W.R.T. A SET OF VIEW ANSWERS

In the example in this appendix, one query is $\mathcal{M}\Sigma$ -conditionally contained in the other, even though the bodies of the queries do not share any relational symbols.

EXAMPLE D.1. *In this trivial example, one query is $\mathcal{M}\Sigma$ -conditionally contained in the other (in the absence of dependencies), even though the bodies of the queries do not share any relational symbols. Consider Boolean CQ queries Q_1 and Q_2 , a CQ view V , and a set of view answers MV , as follows.*

$$\begin{aligned} Q_1() &\leftarrow P(X). \\ Q_2() &\leftarrow R(Y). \\ V(Y) &\leftarrow R(Y). \\ MV &= \{ V(c) \}. \end{aligned}$$

Let $\mathcal{M}\Sigma$ be $(\{P, R\}, \emptyset, \{V\}, MV)$, with V and MV as above. P and R in the schema $\mathbf{P} = \{P, R\}$ are unary relation symbols, and no dependencies hold on \mathbf{P} .

For any base instance I that is relevant to the setting $\mathcal{M}\Sigma$, the instance I must have the ground atom $R(c)$. (This follows from the definitions of the view V and of the instance MV .) As a result, the query Q_2 returns the empty tuple on any such instance I . It follows that any Boolean query, including Q_1 , is $\mathcal{M}\Sigma$ -conditionally contained in Q_2 . The algorithm reported in this paper allows us to make this correct conclusion. \square

E. A NONCONTAINMENT EXAMPLE

In this appendix we show by example that when $Q_1 \sqsubseteq_{\mathcal{M}\Sigma} Q$ holds for some choice of Q_1 , Σ , MV , and Q , then none of the following necessarily holds:

- (1) $Q_1 \sqsubseteq Q$,
- (2) $Q_1 \sqsubseteq_\Sigma Q$, and
- (3) $Q_1 \sqsubseteq_{\mathcal{M}\emptyset} Q$.

Here, by $\mathcal{M}\emptyset$ we denote the result of replacing Σ by \emptyset in the given setting $\mathcal{M}\Sigma = (\mathbf{P}, \Sigma, \mathcal{V}, MV)$.

EXAMPLE E.1. *Recall the schema $\mathbf{P} = \{\text{Emp}, \text{HQDept}, \text{OfficeInHQ}\}$ of Example 1.1. We abbreviate each relation name by its first letter (same as in Example 4.1). As before, we assume that the only primary key of the relation E is its three attributes together. The key of the relation O is its first attribute, which we express using the following egd τ :*

$$\tau : O(X, Y) \wedge O(X, Z) \rightarrow Y = Z.$$

Suppose that for all the departments located in the company headquarters, all their employees have their offices in the headquarters. We express this constraint using a tgd σ (which is the same as in Examples 1.1 and 4.1):

$$\sigma : E(X, Y, Z) \wedge H(Y) \rightarrow \exists S O(X, S).$$

We assume that σ and τ constitute all the integrity constraints Σ on the schema \mathbf{P} , that is, $\Sigma = \{\sigma, \tau\}$.

Recall the views U , V , and W introduced in Example 1.1:

```

(U): DEFINE VIEW U(Dept) AS SELECT * FROM HQDept;
(V): DEFINE VIEW V(Name, Dept) AS
    SELECT DISTINCT Name, Dept FROM Emp;
(W): DEFINE VIEW W(Dept, Salary) AS
    SELECT DISTINCT Dept, Salary FROM Emp;

```

We denote by \mathcal{V} the set $\{U, V, W\}$.

Suppose that a user, or several users together, are authorized to see the answers to all three views U , V , and W , and that at some point in time the user(s) can see the following set MV of answers to these views (same as in Example 1.1).

$MV = \{U(\text{sales}), V(\text{johnDoe}, \text{sales}), W(\text{sales}, 50000)\}$.

We denote by $\mathcal{M}\Sigma$ the materialized-view setting $(\mathbf{P}, \Sigma, \mathcal{V}, MV)$.

Now recall the secret query Q of Example 1.1; Q returns the names and salaries of all the employees who work in the company headquarters.

```

(Q): SELECT DISTINCT E.Name, Salary FROM Emp E, OfficeInHQ
    WHERE E.Name = OfficeInHQ.Name,

```

Recall the tuple $\bar{t} = (\text{johnDoe}, 50000)$ and the query R_{vw} , over the schema \mathcal{V} , of Example A.1:

```

(Rvw): SELECT DISTINCT Name, Salary FROM V, W
    WHERE V.Name = 'johnDoe' AND V.Dept = W.Dept
    AND V.Dept = 'sales' AND Salary = '50000';

```

Using the results of this paper, we can show that the expansion R_{vw}^{exp} , in terms of the schema \mathbf{P} , of the query R_{vw} is contained in the query Q w.r.t. the setting $\mathcal{M}\Sigma$. At the same time, none of the following containments hold: $R_{vw}^{exp} \sqsubseteq Q$, $R_{vw}^{exp} \sqsubseteq_{\Sigma} Q$, and $R_{vw}^{exp} \sqsubseteq_{\mathcal{M}\emptyset} Q$. (Here, by $\mathcal{M}\emptyset$ we denote the result of replacing Σ by \emptyset in the setting $\mathcal{M}\Sigma$.)

We now prove all the containment and non-containment statements of the preceding paragraph, for the queries R_{vw}^{exp} and Q over the schema \mathbf{P} . First, we render in Datalog the queries Rvw , R_{vw}^{exp} , Q , and the queries for the three views. (For conciseness, in the remainder of this example we will refer to the constants johnDoe , sales , and 50000 as c , d , and f , respectively.)

```

Q(X, Z) ← E(X, Y, Z), O(X, S).
U(X)     ← H(X).
V(X, Y)  ← E(X, Y, Z).
W(Y, Z)  ← E(X, Y, Z).
Rvw(c, f) ← V(c, d), W(d, f).
Rvw^{exp}(c, f) ← E(c, d, Z), E(X, d, f).

```

(1) We are first determining whether $R_{vw}^{exp} \sqsubseteq Q$ holds. The noncontainment of $R_{vw}^{exp}(c, f)$ in Q is immediate from the containment test of [9] and from the absence in the definition of $R_{vw}^{exp}(c, f)$ of a subgoal with predicate OfficeInHQ . (As a result, the subgoal of Q with predicate OfficeInHQ cannot be mapped into the body of $R_{vw}^{exp}(c, f)$.) We conclude that $R_{vw}^{exp} \not\sqsubseteq Q$ does not hold.

(2) We are now determining whether $R_{vw}^{exp} \sqsubseteq_{\Sigma} Q$ holds. We observe that the result of chasing the query $R_{vw}^{exp}(c, f)$ with the dependencies Σ is identical to $R_{vw}^{exp}(c, f)$. Recall that $R_{vw}^{exp}(c, f) \sqsubseteq_{\Sigma} Q$ holds iff that chase result (which is identical to $R_{vw}^{exp}(c, f)$) is contained in Q in the absence of dependencies. We then use the reasoning of item (1) to conclude that $R_{vw}^{exp} \not\sqsubseteq_{\Sigma} Q$ does not hold.

(3) We are now determining whether $R_{vw}^{exp} \sqsubseteq_{\mathcal{M}\emptyset} Q$ holds. Consider the following instance I of schema \mathbf{P} :

$I = \{H(d), E(c, d, f)\}$.

It is easy to verify that for the set $\mathcal{V} = \{U, V, W\}$, the result of applying the queries for \mathcal{V} to I is exactly the instance MV as given above. (Observe that the instance I does not satisfy the $\text{tgd } \sigma$ in the set Σ in the setting $\mathcal{M}\Sigma$. At the same time, we're checking here for the containment of R_{vw}^{exp} in Q w.r.t. the setting $\mathcal{M}\emptyset$, in which $\Sigma = \emptyset$.) We verify that $Q(I) = \emptyset$ and that $R_{vw}^{exp}(c, f)(I) = \{(c, f)\}$. As a result, the containment $R_{vw}^{exp} \sqsubseteq_{\mathcal{M}\emptyset} Q$ does not hold.

(4) Finally, let us determine whether $R_{vw}^{exp} \sqsubseteq_{\mathcal{M}\Sigma} Q$ holds. For ease of exposition, we denote the query R_{vw}^{exp} by Q_1 . Using the results of this current paper, we first transform Q_1 into Q'_1 , by conjoining the body of Q_1 with $C_{MV}^{exp} = E(c, d, A) \wedge E(B, d, f) \wedge H(d)$. (We then minimize the resulting query to obtain Q'_1 ; the minimization does not affect any of our results in this current paper.)

```

Q_1(c, f)  ← E(c, d, Z), E(X, d, f).
Q'_1(c, f) ← E(c, d, Z), E(X, d, f), H(d).

```

Then we chase Q'_1 using the MV -induced dependencies τ_U , τ_V , and τ_W , as well as the modifications σ' and τ' of the dependencies σ and τ , respectively, in the given set Σ of dependencies on the schema \mathbf{P} :

```

τ_U : H(X)           → X = d.
τ_V : E(X, Y, Z)      → X = c ∧ Y = d.
τ_W : E(X, Y, Z)      → Y = d ∧ Z = f.
σ' : E(X, Y, Z) ∧ H(T) → (∃S O(X, S)) ∨ (Y ≠ T).
τ' : O(X, Y) ∧ O(T, Z) → (Y = Z) ∨ (X ≠ T).

```

The result of the chase of Q'_1 with the dependencies τ_U , τ_V , τ_W , σ' , and τ' is the following query $(Q_1)^{\mathcal{M}\Sigma}$:

$(Q_1)^{\mathcal{M}\Sigma}(c, f) \leftarrow E(c, d, f), H(d), O(c, Z).$

It is easy to verify that, by the results of [9], the CQ query $(Q_1)^{\mathcal{M}\Sigma}$ is unconditionally contained in the input query Q . Using the results of this current paper, we obtain that $Q_1 \sqsubseteq_{\mathcal{M}\Sigma} Q$ holds as well. On replacing Q_1 by the original notation R_{vw}^{exp} , we conclude that $R_{vw}^{exp} \sqsubseteq_{\mathcal{M}\Sigma} Q$ also holds. \square

F. FOR $\mathcal{M}\Sigma$ -CONDITIONAL CONTAINMENT, CANNOT JUST CHASE WITH Σ (OR EVEN WITH $\Sigma_{(\neq)}$) THE QUERY Q''_1

This appendix illustrates via an example that, in determining $\mathcal{M}\Sigma$ -conditional containment of two CQ queries w.r.t. a weakly acyclic materialized-view setting, just “chasing with Σ (or even with $\Sigma_{(\neq)}$)” the query Q''_1 of [31] may yield incorrect conclusions about the $\mathcal{M}\Sigma$ -conditional containment of the input queries. (“The query Q''_1 of [31]” is the result of chasing one of the input queries in the algorithm of [31] for determining conditional containment of the two input queries in the absence of dependencies. The algorithm of [31] checks this query Q''_1 for unconditional containment in the input query Q_2 ; it is shown in [31] that, in case of the positive answer to the unconditional-containment test, the input query Q_1 is *conditionally* contained in Q_2 .)

Note. As shown in this current paper, the result $(Q_1)^{\mathcal{M}\Sigma}$ of chasing a CQ query Q_1 using weakly acyclic

dependencies Σ and the MV -induced dependencies is, in general, a UCQ^\neq query. $(Q_1)^{\mathcal{M}\Sigma}$ in Example F.1 is a CQ query, because the view/tgd definitions and MV are particularly simple here.

EXAMPLE F.1. Let schema \mathbf{P} have a unary relation symbol R and binary relation symbols P and S . Consider a tgdt σ and three view definitions:

$$\begin{aligned}\sigma : S(X, Y) &\rightarrow \exists T \ P(Y, T). \\ U(X) &\leftarrow R(X). \\ V(X) &\leftarrow P(Y, X). \\ W(X) &\leftarrow S(Y, X).\end{aligned}$$

Finally, for the set $\{U, V, W\}$ of the above views, let the set of view answers MV be $\{U(c), V(c), W(f)\}$. Then the materialized-view setting $(\mathbf{P}, \{\sigma\}, \{U, V, W\}, MV)$, which we denote by $\mathcal{M}\Sigma$, is CQ weakly acyclic.

Now let Q_1 and Q_2 be two CQ queries, as follows.

$$\begin{aligned}Q_1(X) &\leftarrow S(X, Y). \\ Q_2(X) &\leftarrow S(X, Y), P(Y, Z), R(Z).\end{aligned}$$

It is easy to show that neither $Q_1 \sqsubseteq Q_2$ nor $Q_1 \sqsubseteq_{\{\sigma\}} Q_2$ holds. (Please see Section 3.) However, it turns out that $Q_1 \sqsubseteq_{\mathcal{M}\Sigma} Q_2$ does hold. A way to prove this fact is to chase the query Q_1 using both the given dependencies, $\{\sigma\}$, on the schema \mathbf{P} , and the “ MV -induced” dependencies that we introduce in this paper. (In this particular example, the input dependency σ is the same as its “neq-transformation.” That is, the set of dependencies $\Sigma_{(\neq)}$ required for the chase in our approach for checking $\mathcal{M}\Sigma$ -conditional containment, is the same as the input set $\Sigma = \{\sigma\}$. Thus, chase with σ and with the MV -induced dependencies as shown in this example is correct w.r.t. the approach for checking $\mathcal{M}\Sigma$ -conditional containment as introduced in this paper.)

The result of the chase is the following CQ query:

$$(Q_1)^{\mathcal{M}\Sigma}(X) \leftarrow S(X, f), P(f, c), R(c), P(Z, c), S(T, f).$$

We can then determine that the $\mathcal{M}\Sigma$ -conditional containment of Q_1 in Q_2 holds, by using the results of [9] to check that the unconditional containment $(Q_1)^{\mathcal{M}\Sigma} \sqsubseteq Q_2$ holds.

We now provide the details of chasing the query Q_1 using both the given dependencies, $\{\sigma\}$, on the schema \mathbf{P} , and the MV -induced dependencies. The process of obtaining the query $(Q_1)^{\mathcal{M}\Sigma}$ from the query Q_1 via this process can be represented using four stages, as follows:

Stage I: We first add to the body of the query Q_1 the conjunction $\mathcal{C}_{MV}^{exp} = R(c) \wedge P(Z, c) \wedge S(T, f)$:

$$Q'_1(X) \leftarrow S(X, Y), R(c), P(Z, c), S(T, f).$$

Stage II: In this stage, we choose to chase the query Q'_1 using the following MV -induced dependencies τ_U , τ_V , and τ_W . (Intuitively, each of the egds τ_U , τ_V , and τ_W below arises from the ground fact for the respective view in the given instance MV . If MV had more than one fact for any view symbol, call it V^* , then all these facts together would give rise to a single dependency for V^* , with a disjunction on the right-hand side of the dependency. The reason each MV -induced dependency in this example is “just” an egd is that both the definitions of the views and the instance MV are particularly simple here.)

$$\begin{aligned}\tau_U : R(X) &\rightarrow X = c. \\ \tau_V : P(Y, X) &\rightarrow X = c. \\ \tau_W : S(Y, X) &\rightarrow X = f.\end{aligned}$$

Applying these three dependencies to the query Q'_1 results in the following query:

$$Q''_1(X) \leftarrow S(X, f), R(c), P(Z, c), S(T, f).$$

The difference between the queries Q'_1 and Q''_1 is that an application of the egd τ_W to the first subgoal, $S(X, Y)$, of the query Q'_1 turns this subgoal into the first subgoal $S(X, f)$ of the query Q''_1 . None of the dependencies τ_U , τ_V , and τ_W is applicable to the query Q''_1 .

Stage III: In this stage, we choose to chase the query Q''_1 with the dependency σ (on the schema \mathbf{P} in $\mathcal{M}\Sigma$); the outcome of the chase is a CQ query Q'''_1 , see below. The two chase steps with σ on Q''_1 result in the addition to the body of Q''_1 of two subgoals, $P(f, A)$ and $P(f, B)$. These two subgoals are what is different between the queries Q''_1 and Q'''_1 . The tgdt σ is not applicable to the query Q'''_1 .

$$Q'''_1(X) \leftarrow S(X, f), R(c), P(Z, c), S(T, f), P(f, A), P(f, B).$$

Note that if we stop here, unconditional containment of the query Q'''_1 in Q_2 does not hold. (The reason is, the body of Q'''_1 does not have any pattern of three subgoals with predicates S , P , and R , such that the body of the query Q_2 would subsume the pattern.)

Stage IV: Now, after stage III in chasing the original query Q_1 is over, we can apply again chase steps with the MV -induced dependencies. Indeed, we can do two chase steps with the egd τ_V . As a result, the subgoals $P(f, A)$ and $P(f, B)$ of the query Q'''_1 are transformed into two identical atoms $P(f, c)$. We call the resulting query

$$(Q_1)^{\mathcal{M}\Sigma}(c) \leftarrow S(X, f), P(f, c), R(c), P(Z, c), S(T, f).$$

Chase with σ or with the MV -induced dependencies does not apply to this query $(Q_1)^{\mathcal{M}\Sigma}$. It is easy to see that the query $(Q_1)^{\mathcal{M}\Sigma}$ is unconditionally contained in the query Q_2 . We conclude that the given query Q_1 is $\mathcal{M}\Sigma$ -conditionally contained in Q_2 . \square

G. ENSURING THAT THE IMAGE OF $BODY((Q_1)^{\mathcal{M}\Sigma})$ UNDER EACH VALUATION BE A Σ -VALID BASE INSTANCE FOR \mathcal{V} AND MV

In this appendix, in the context of the problem of determining $\mathcal{M}\Sigma$ -conditional query containment, we demonstrate via two examples the role of disequality atoms, both in MV -induced dependencies (Example G.1) and in the dependencies $\Sigma_{(\neq)}$ (Example G.2). As a summary of the observations illustrated by these examples, when at least some of the disequalities are missing in either kind of dependencies, then there may exist a valuation, call it ν , from the body, B , of the resulting query to some instance, such that the image of (the relational part of) B under ν is not a Σ -valid base instance for \mathcal{V} and MV . As a result, we would not be able to prove correctness of the algorithm for determining $\mathcal{M}\Sigma$ -conditional containment of CQ queries w.r.t.

CQ weakly acyclic materialized-view settings, both in case $\Sigma = \emptyset$ (this is the setting of [31]) and in case $\Sigma \neq \emptyset$. (Example G.1 illustrates the former case, and Example G.2 – the latter case.) Specifically, we would not be able to prove the claim that $Q_1 \sqsubseteq_{\mathcal{M}\Sigma} Q_2$ implies $(Q_1)^{\mathcal{M}\Sigma} \sqsubseteq Q_2$.

EXAMPLE G.1. On a schema \mathbf{P} with one binary relation symbol P , consider two CQ queries, Q_1 and V , as follows:

$$Q_1(X) \leftarrow P(X, Y). \\ V(X) \leftarrow P(X, X).$$

Let \mathcal{V} be the set $\{V\}$, with the set of view answers $MV = \{V(c)\}$. We will show chase of the query Q_1 with the dependencies arising in the setting $\mathcal{M}\Sigma = (\mathbf{P}, \emptyset, \mathcal{V}, MV)$.

Scenario A. In this scenario, we show the correct chase of the query Q_1 for the setting $\mathcal{M}\Sigma$, as introduced in [31]. The first step of the approach is to conjoin the body of Q_1 with the $\mathcal{C}_{MV}^{\text{exp}} = P(c, c)$:

$$Q'_1(X) \leftarrow P(X, Y), P(c, c).$$

We then chase the query Q'_1 with the MV -induced dependency τ_V :

$$\tau_V : P(X, Y) \rightarrow X = c \vee X \neq Y.$$

The result of the chase of Q'_1 with τ_V is a UCQ^\neq query $(Q_1)^{\mathcal{M}\Sigma} = \{Q_1^{(a)}, Q_1^{(b)}\}$, with the CQ^\neq components specified as follows:

$$Q_1^{(a)}(c) \leftarrow P(c, Y), P(c, c). \\ Q_1^{(b)}(X) \leftarrow P(X, Y), X \neq Y, P(c, c).$$

We can show that for each instance I of the schema \mathbf{P} and for each valuation, ν , for either one of the two CQ^\neq components of the query $(Q_1)^{\mathcal{M}\Sigma}$ and for I , the image under ν of (the relational part of) the body of the relevant CQ^\neq component of $(Q_1)^{\mathcal{M}\Sigma}$ is a Σ -valid base instance for \mathcal{V} and MV . (Here, $\Sigma = \emptyset$ as specified above, and \mathcal{V} and MV are also as above.)

Scenario B. In this scenario, we show chase of the query Q_1 using a version of the dependency τ_V (which is defined as in Scenario A here) that does not use disequalities. We refer to this version of τ_V as $\tilde{\tau}_V$:

$$\tilde{\tau}_V : P(X, X) \rightarrow X = c.$$

(Recall that the body of the view V is $P(X, X)$.)

Similarly to Scenario A, we first obtain the query Q'_1 , and then chase it with $\tilde{\tau}_V$. The result of chase of Q'_1 with $\tilde{\tau}_V$ is a CQ query \tilde{Q}_1 specified as follows:

$$\tilde{Q}_1(X) \leftarrow P(X, Y), P(c, c).$$

As the dependency $\tilde{\tau}_V$ does not apply to (either Q'_1 or) \tilde{Q}_1 , the query \tilde{Q}_1 is the result of the chase of the query Q'_1 with $\tilde{\tau}_V$.

Consider an instance $I = \{P(c, c), P(d, d)\}$ of the schema \mathbf{P} , and the valuation $\nu : \{X \rightarrow d, Y \rightarrow d\}$ for \tilde{Q}_1 and I . Clearly, the image $J = \{P(d, d), P(c, c)\}$ of

(the relational part of) the body of the query \tilde{Q}_1 under ν is an instance (of schema \mathbf{P}) that does not generate the above set of view answers MV under CWA. That is, J is not a \emptyset -valid base instance for the $\{V\}$ and MV as above. The reason is, the relation $V(J)$ has the tuple $V(d)$, which is not in the instance MV as specified in the beginning of this example. \square

EXAMPLE G.2. On a schema \mathbf{P} with a binary relation symbol P and a unary relation symbol S , consider two CQ queries, Q_1 and V , and a tgd σ , all defined as follows:

$$Q_1(X) \leftarrow P(X, Y). \\ V(X) \leftarrow P(X, Y). \\ \sigma : P(X, X) \rightarrow S(X)$$

Let \mathcal{V} be the set $\{V\}$, with the set of view answers $MV = \{V(c)\}$. We will show chase of the query Q_1 with the dependencies arising in the setting $\mathcal{M}\Sigma = (\mathbf{P}, \{\sigma\}, \mathcal{V}, MV)$.

Scenario A. In this scenario, we show the correct chase of the query Q_1 for the setting $\mathcal{M}\Sigma$, as introduced in this current paper. The first step of the approach is to conjoin the body of Q_1 with the $\mathcal{C}_{MV}^{\text{exp}} = P(c, c)$:

$$Q'_1(X) \leftarrow P(X, Y), P(c, c).$$

We then chase the query Q'_1 with the MV -induced dependency τ_V and with the “neq-transformation” σ' of the tgd σ , defined as follows:

$$\tau_V : P(X, Y) \rightarrow X = c \\ \sigma' : P(X, Y) \rightarrow S(X) \vee X \neq Y$$

The result of the chase of Q'_1 with τ_V and σ' is a UCQ^\neq query $(Q_1)^{\mathcal{M}\Sigma} = \{Q_1^{(a)}, Q_1^{(b)}, Q_1^{(c)}, Q_1^{(d)}\}$, with the CQ^\neq components specified as follows:

$$Q_1^{(a)}(c) \leftarrow P(c, Y), S(c), P(c, c). \\ Q_1^{(b)}(c) \leftarrow P(c, Y), S(c), P(c, Z), Z \neq c. \\ Q_1^{(c)}(c) \leftarrow P(c, Y), Y \neq c, P(c, Z), S(c). \\ Q_1^{(d)}(c) \leftarrow P(c, Y), Y \neq c, P(c, Z), Z \neq c.$$

We can show that for each instance I of the schema \mathbf{P} and for each valuation, ν , for any one of the four CQ^\neq components of the query $(Q_1)^{\mathcal{M}\Sigma}$ and for I , the image under ν of (the relational part of) the body of the relevant CQ^\neq component of $(Q_1)^{\mathcal{M}\Sigma}$ is a Σ -valid base instance for \mathcal{V} and MV . (Here, $\Sigma = \{\sigma\}$ as specified above, and \mathcal{V} and MV are also as above.)

Scenario B. In this scenario, we show chase of the query Q_1 using the dependency τ_V (defined as in Scenario A here), as well as the original tgd σ (which does not use disequalities), instead of using the dependency σ' of Scenario A.

Similarly to Scenario A, we first obtain the query Q'_1 , and then chase it with τ_V and σ . The result of the chase is a CQ query \tilde{Q}_1 specified as follows:

$$\tilde{Q}_1(c) \leftarrow P(c, Y), P(c, Z).$$

Neither τ_V nor σ applies to \tilde{Q}_1 .

Consider an instance $I = \{P(c, c), S(c)\}$ of the schema \mathbf{P} , and the valuation $\nu : \{Y \rightarrow c, Z \rightarrow c\}$ for \tilde{Q}_1 and I .

Clearly, the image $J = \{P(c, c)\}$ of (the relational part of) the body of the query \bar{Q}_1 under ν is an instance (of schema \mathbf{P}) that does not satisfy the input $\text{tgd } \sigma$ (even though the instance I does). (The instance J does generate the above set of view answers MV under CWA.) We conclude that J is not a Σ -valid base instance for the $\{V\}$ and MV as above. \square

H. THE DATA-EXCHANGE APPROACH

In this appendix we outline an approach to finding the set of certain answers to a CQ query w.r.t. a CQ weakly acyclic materialized-view setting under CWA. (Please see Appendix I for all the technical details.) This approach is based on data exchange [13, 5, 4], hence the name.

This approach is the result of our having rediscovered independently the idea and methods of the 2005 paper [27] by Stoffel and colleagues. The work [27] explicitly uses techniques that arise in data exchange, to solve the problem of finding the set of certain answers to a query w.r.t. a materialized-view setting under the *open-world assumption* (OWA). At the same time, Brodsky and colleagues in their paper [8], which was published in 2000, used the same approach as Stoffel and colleagues did in [27], without calling their approach (of [8]) “data exchange.” (Arguably, “data exchange” was not a household term in the year 2000.) Both [8] and [27] solve the problem of finding the set of certain answers to a query w.r.t. a materialized-view setting under the open-world assumption (OWA). (Please see Section 2 for the details on the query languages and classes of dependencies to which the work of [8] and [27] applies.)

In this Appendix H we show that, not surprisingly, the approach of [8] and [27] is sound but not complete under the closed-world assumption (CWA), even in case when the given (base) schema comprises a single relation, and even in the absence of dependencies on this schema. (A counterexample can be found in Appendix A.) Our “view-verified data exchange” of Appendix J then provides a correct algorithm for solving the problem of finding the set of certain answers to a query w.r.t. a materialized-view setting under CWA, for CQ queries and CQ weakly acyclic materialized-view settings.

The idea of using data exchange [13, 5, 4] as a tool arises naturally in the context of the problem of finding the set of certain query answers w.r.t. a materialized-view setting. In the remainder of this appendix, we outline the resulting “data-exchange” approach. We begin by reviewing the basics of data exchange in Section H.1, by generally following the excellent detailed survey [5]. Then, in Section H.2 we introduce and discuss the sound but not complete data-exchange approach to finding the set of certain answers to a CQ query w.r.t. a CQ weakly acyclic materialized-view setting under CWA. All the technical details of the discussion can be found in Appendix I.

H.1 Reviewing Data Exchange

Given schemas $\mathbf{S} = \langle S_1, \dots, S_m \rangle$ and $\mathbf{T} = \langle T_1, \dots, T_n \rangle$, with no relation symbols in common, denote by $\langle \mathbf{S}, \mathbf{T} \rangle$ the schema $\langle S_1, \dots, S_m, T_1, \dots, T_n \rangle$. If I is an instance of \mathbf{S} and J an instance of \mathbf{T} , then (I, J) denotes an instance K of $\langle \mathbf{S}, \mathbf{T} \rangle$ such that $K[S_i] = I[S_i]$ and $K[T_j] = J[T_j]$, for $i \in [1, m]$ and $j \in [1, n]$.

DEFINITION H.1. (Data-exchange setting) A data-

exchange setting \mathcal{M} is a triple $(\mathbf{S}, \mathbf{T}, \Sigma)$, where \mathbf{S} and \mathbf{T} are disjoint schemas and Σ is a finite set of dependencies over $\langle \mathbf{S}, \mathbf{T} \rangle$. \mathbf{S} in \mathcal{M} is called the source schema, and \mathbf{T} is called the target schema. \square

Instances of \mathbf{S} are called *source* instances and are always ground instances. Instances of \mathbf{T} are *target* instances. Given a source instance I , we say that a target instance J is a *solution* for I (under \mathcal{M}) if $(I, J) \models \Sigma$.

It is customary in the data-exchange literature to restrict the study to the class of settings whose set Σ can be split into two sets Σ_{st} and Σ_t , as follows:

1. Σ_{st} is a set of *source-to-target* dependencies (*stds*), that is, tgds of the form $\varphi_{\mathbf{S}}(\bar{X}) \rightarrow \exists \bar{y} \psi_{\mathbf{T}}(\bar{X}, \bar{Y})$, where $\varphi_{\mathbf{S}}(\bar{X})$ and $\psi_{\mathbf{T}}(\bar{X}, \bar{Y})$ are conjunctions of relational atoms in \mathbf{S} and \mathbf{T} , respectively; and
2. Σ_t , the set of *target* dependencies, is the union of a set of tgds and egds defined over the schema \mathbf{T} .

In this current paper, we assume all data-exchange settings to be of the form $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where $\Sigma = \Sigma_{st} \cup \Sigma_t$, for Σ_{st} a set of stds and Σ_t a set of target dependencies. Intuitively, the stds can be viewed as a tool for specifying how the source data get translated into target data. In addition, the target dependencies are the usual database constraints, to be satisfied by the translated data. The data-exchange settings of this form are not restrictive from the database point of view.

Solutions for a given source instance are not necessarily unique, and there are source instances that have no solutions. *Universal solutions* are, intuitively, “the most general” solutions among all possible solutions. Formally, given a solution J for source instance I , we say that J is a *universal* solution for I if for every solution J' for I , there exists a homomorphism from J to J' . Constructing a universal solution for a given source instance I can be done by chasing I with $\Sigma_{st} \cup \Sigma_t$. The chase may never terminate or may fail; in the latter case, no solution exists [13]. If the chase does not fail and terminates, then the resulting target instance is guaranteed to be a universal solution for I .

The problem of checking for the existence of solutions is known to be undecidable, please see [5]. At the same time, the following positive result is due to [13].

THEOREM H.1. [13] *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t)$ be a fixed data-exchange setting, such that Σ_t is weakly acyclic. Then there is a polynomial-time algorithm such that for every source instance I , the algorithm decides whether a solution for I exists. Then, whenever a solution for I exists, the algorithm computes a universal solution for I in polynomial time.* \square

The universal solution of Theorem H.1, called the *canonical* universal solution [13], is the result of the chase.

Query answering: Assume that a user poses a query Q over the target schema \mathbf{T} , and I is a given source instance. Then the usual semantics for the query answering is that of “certain answers,” defined as follows. Let \mathcal{M} be a data-exchange setting, let Q be a query over the target schema \mathbf{T} of \mathcal{M} , and let I be a source instance. We define *certain* $_{\mathcal{M}}(Q, I)$, the set of *certain answers* of Q with respect to I under \mathcal{M} , as

$$\text{certain}_{\mathcal{M}}(Q, I) = \bigcap \{ Q(J) \mid J \text{ is a solution for } I \}.$$

Computing certain answers for arbitrary FO queries is an undecidable problem. For unions of CQ queries (UCQ queries) we have the following positive result:

THEOREM H.2. [13] *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t)$ be a data-exchange setting with Σ_t a weakly acyclic set, and let Q be a UCQ query. Then the problem of computing certain answers for Q under \mathcal{M} can be solved in polynomial time.* \square

To compute the certain answers to a UCQ query Q w.r.t. a source instance I , we first check whether a solution for I exists. If there is no solution, the setting is inconsistent w.r.t. I . Otherwise, compute an arbitrary universal solution J for I , and then compute the set $Q_{\downarrow}(J)$ of all those tuples in $Q(J)$ that do not contain nulls. It can be shown that $Q_{\downarrow}(J) = \text{certain}_{\mathcal{M}}(Q, I)$.

H.2 Data Exchange for Finding Certain Query Answers w.r.t. Materialized-View Setting

Suppose we are given a valid CQ weakly acyclic materialized-view setting $\mathcal{M}\Sigma = (\mathbf{P}, \Sigma, \mathcal{V}, MV)$ and a CQ query Q of arity $k \geq 0$. We consider the problem of finding the set of certain answers to Q w.r.t. the setting $\mathcal{M}\Sigma$ under CWA. That is, by the definition given in Section 4.1, we are interested in finding all (and only) the k -ary tuples \bar{t} of elements of $\text{consts}(\mathcal{M}\Sigma)$, such that for all the instances I with $\mathcal{V} \Rightarrow_{I, \Sigma} MV$, we have $\bar{t} \in Q(I)$.

In this subsection we show how a straightforward reformulation of the pair $(\mathcal{M}\Sigma, Q)$ turns the above problem into an instance of the problem of computing certain answers in data exchange. We first construct a set Σ_{st} of tgds, as follows. For a view V in the set of views \mathcal{V} in $\mathcal{M}\Sigma$, consider the query $V(\bar{X}) \leftarrow \text{body}_{(V)}(\bar{X}, \bar{Y})$ for V . (As $\mathcal{M}\Sigma$ is a CQ setting, the query for each $V \in \mathcal{V}$ is a CQ query.) We associate with this $V \in \mathcal{V}$ the tgd $\sigma_V : V(\bar{X}) \rightarrow \exists \bar{Y} \text{body}_{(V)}(\bar{X}, \bar{Y})$. We then define the set Σ_{st} to be the set of tgds σ_V for all $V \in \mathcal{V}$. Then the components \mathbf{P} , Σ , and \mathcal{V} of $\mathcal{M}\Sigma$ can be reformulated into the following data-exchange setting:

$$\mathcal{S}^{(de)}(\mathcal{M}\Sigma) = (\mathcal{V}, \mathbf{P}, \Sigma_{st} \cup \Sigma).$$

Further, we interpret MV in $\mathcal{M}\Sigma$ as a source instance for $\mathcal{S}^{(de)}(\mathcal{M}\Sigma)$, and interpret the input query Q as a query on the target schema \mathbf{P} in $\mathcal{S}^{(de)}(\mathcal{M}\Sigma)$. We call the triple $(\mathcal{S}^{(de)}(\mathcal{M}\Sigma), MV, Q)$ the associated data-exchange instance for $(\mathcal{M}\Sigma, Q)$.

For valid CQ weakly acyclic settings $\mathcal{M}\Sigma$ and for CQ queries Q , we introduce the following algorithm, which we call the *data-exchange approach to finding the set of certain query answers w.r.t. a materialized-view setting*. First, we compute the canonical universal solution, $J_{de}^{\mathcal{M}\Sigma}$, for the source instance MV in the data-exchange setting $\mathcal{S}^{(de)}(\mathcal{M}\Sigma)$. If $J_{de}^{\mathcal{M}\Sigma}$ does not exist, then we output the empty set of answers. Otherwise we output, as a set of certain answers to the query Q w.r.t. the setting $\mathcal{M}\Sigma$, the set of all those tuples in $Q(J_{de}^{\mathcal{M}\Sigma})$ that do not contain nulls. When we assume, similarly to [31], that everything in $\mathcal{M}\Sigma$ is fixed except for MV and Q , then from Theorem H.2 due to [13] we obtain immediately that this algorithm always terminates and runs in polynomial time. We have shown that this data-exchange approach is sound. (Please see Appendix I.)

It turns out that our data-exchange approach is not complete for (CQ queries and) CQ weakly acyclic settings $\mathcal{M}\Sigma$ with $\Sigma = \emptyset$, nor for those with $\Sigma \neq \emptyset$. (Please see Appendix I.2 for all the details.) We now discuss a feature of the data-exchange approach that prevents us from using it as a complete algorithm for the problem of finding the set of certain query answers w.r.t. a materialized-view setting under CWA. In Appendix J we will eliminate this feature of the data-exchange approach, in a modification that will yield a sound and complete algorithm for finding the set of certain answers to CQ queries w.r.t. CQ weakly acyclic materialized-view settings under CWA.

Why is the data-exchange approach not complete when applied to (CQ queries and) CQ weakly acyclic materialized-view settings? Intuitively, the problem is that its canonical universal solution $J_{de}^{\mathcal{M}\Sigma}$ “may cover too many target instances” (i.e., $J_{de}^{\mathcal{M}\Sigma}$ is an OWA rather than CWA solution). Let us rewrite the set MV of Example A.1 using, to save space, constants c , d , and f , as $MV = \{V(c, d), W(d, f)\}$. Now let us evaluate the queries for the views V and W of Example A.1 over the canonical solution $J_{de}^{\mathcal{M}\Sigma} = \{E(c, d, \perp_1), E(\perp_2, d, f)\}$ for that example. We obtain that the answer to the view V on $J_{de}^{\mathcal{M}\Sigma}$ is $\{V(c, d), V(\perp_2, d)\}$. Similarly, the answer to W on $J_{de}^{\mathcal{M}\Sigma}$ is $\{W(d, \perp_1), W(d, f)\}$. Thus, if we replace \perp_1 in $J_{de}^{\mathcal{M}\Sigma}$ by any constant except f , or replace \perp_2 by any constant except c , then any ground instance obtained from $J_{de}^{\mathcal{M}\Sigma}$ using these replacements would “generate too many tuples” (as compared with MV) in the answer to either V or W .

We now generalize over this observation. Fix a valid CQ weakly acyclic instance $\mathcal{M}\Sigma$, and consider the canonical universal solution (if one exists) $J_{de}^{\mathcal{M}\Sigma}$ generated by the data-exchange approach with $\mathcal{M}\Sigma$ as input. (In the remainder of this paper, we will refer to $J_{de}^{\mathcal{M}\Sigma}$ as the *canonical data-exchange solution for $\mathcal{M}\Sigma$* .) By definition of $J_{de}^{\mathcal{M}\Sigma}$, for each $V \in \mathcal{V}$, the answer to the query for V on $J_{de}^{\mathcal{M}\Sigma}$ is a superset of the relation $MV[V]$. Suppose that the answer on $J_{de}^{\mathcal{M}\Sigma}$ to at least one view $V \in \mathcal{V}$ is not a subset of $MV[V]$, as it is the case in the example we have just discussed. Then $J_{de}^{\mathcal{M}\Sigma}$, as a template for instances of schema \mathbf{P} , describes not only instances that “generate” exactly the set MV in $\mathcal{M}\Sigma$, but also those instances that generate proper supersets of MV . The latter instances are not of interest to us. (Recall that we take the CWA viewpoint, and thus are interested only in the instances I of schema \mathbf{P} such that $\mathcal{V} \Rightarrow_{I, \Sigma} MV$.) As a result, when the data-exchange approach uses $J_{de}^{\mathcal{M}\Sigma}$ to obtain certain answers to the input query Q , it can easily miss those certain answers that characterize only those instances of interest to us.

I. TECHNICAL DETAILS OF THE DATA-EXCHANGE APPROACH

In this appendix we discuss the technical details of the data-exchange approach of Appendix H to finding the set of certain answers to a query w.r.t. a materialized-view setting, under CWA for CQ queries and CQ weakly acyclic settings.

I.1 A Sound Data-Exchange Approach

Suppose that we are given a valid CQ materialized-view setting $\mathcal{M}\Sigma = (\mathbf{P}, \Sigma, \mathcal{V}, MV)$ and a CQ query Q of arity $k \geq 0$. By the definition given in Section 4.1, we are interested in finding all (and only) k -ary tuples \bar{t} of elements of $\text{consts}(\mathcal{M}\Sigma)$, such that for all the instances I satisfying $\mathcal{V} \Rightarrow_{I, \Sigma} MV$, we have $\bar{t} \in Q(I)$.

We now show how a straightforward reformulation of $\mathcal{M}\Sigma$ turns the above problem into an instance of the problem of computing certain answers in data exchange. We first construct a set Σ_{st} of tgds, as follows. For a view V in the set of views \mathcal{V} in $\mathcal{M}\Sigma$, consider the query $V(\bar{X}) \leftarrow \text{body}_{(V)}(\bar{X}, \bar{Y})$ for V . (As $\mathcal{M}\Sigma$ is a CQ instance, the query for each V in \mathcal{V} is a CQ query.) We associate with this $V \in \mathcal{V}$ the tgd $\sigma_V : V(\bar{X}) \rightarrow \exists \bar{Y} \text{body}_{(V)}(\bar{X}, \bar{Y})$. We then define the set Σ_{st} to be the set of tgds σ_V for all $V \in \mathcal{V}$. Then $\mathcal{M}\Sigma$ can be reformulated into a data-exchange setting

$$\mathcal{S}^{(de)}(\mathcal{M}\Sigma) = (\mathcal{V}, \mathbf{P}, \Sigma_{st} \cup \Sigma),$$

with a source instance MV and a query Q on the target schema \mathbf{P} . We call the triple $(\mathcal{S}^{(de)}(\mathcal{M}\Sigma), MV, Q)$ the associated data-exchange instance for $\mathcal{M}\Sigma$ and Q .

The following observation is immediate from Definition 4.2 and from the definitions in Section H.1.

PROPOSITION I.1. *Given a valid CQ materialized-view setting $\mathcal{M}\Sigma$ and a CQ query Q , with their associated data-exchange instance $(\mathcal{S}^{(de)}(\mathcal{M}\Sigma), MV, Q)$. Then for each tuple \bar{t} that is a certain answer of Q with respect to MV under the data-exchange setting $\mathcal{S}^{(de)}(\mathcal{M}\Sigma)$, we have that \bar{t} is a certain answer to the query Q w.r.t. $\mathcal{M}\Sigma$ under CWA.* \square

For valid CQ weakly acyclic settings $\mathcal{M}\Sigma$ and CQ queries Q , we introduce the following algorithm, which we call the *data-exchange approach to finding certain query answers w.r.t. a materialized-view setting*. First, we compute the canonical universal solution, $J_{de}^{\mathcal{M}\Sigma}$, for the source instance MV in the data-exchange setting $\mathcal{S}^{(de)}(\mathcal{M}\Sigma)$. If $J_{de}^{\mathcal{M}\Sigma}$ does not exist, then we output the empty set of answers. Otherwise we output, as a set of certain answers to Q w.r.t. $\mathcal{M}\Sigma$, the set of all those tuples in $Q(J_{de}^{\mathcal{M}\Sigma})$ that do not contain nulls. When we assume, same as in [31], that everything in $\mathcal{M}\Sigma$ is fixed except for MV , and assume that Q is not fixed, then from Theorem H.2 due to [13] we obtain immediately that this algorithm always terminates, constructs the instance $J_{de}^{\mathcal{M}\Sigma}$ in polynomial time, and returns each certain-answer tuple in polynomial time. By Proposition I.1, this data-exchange approach is sound.

I.2 The Data-Exchange Approach Is Not Complete

By Theorem B.2 and Proposition I.1, we have that for CQ weakly acyclic materialized-view settings $\mathcal{M}\Sigma$ and for CQ queries Q , for all the certain answers to Q w.r.t. $\mathcal{M}\Sigma$ that can be found using the data-exchange approach of Section I.1, these certain answers can in principle also be disclosed by an adaptation of our rewriting approach of Section B.2. (See Note 2 in Section B.3.)

THEOREM I.1. *Given a valid CQ weakly acyclic materialized-view setting $\mathcal{M}\Sigma$ and a CQ query Q . Let \mathcal{T}*

be the set of tuples output by the data-exchange approach of Section I.1 when it is applied to the inputs $\mathcal{M}\Sigma$ and Q . Then for each $\bar{t} \in \mathcal{T}$, there exists an MV-validated head-instantiated rewriting R for \bar{t} , such that $R \sqsubseteq_{\Sigma, MV, \mathcal{V}} Q$. \square

Even in the light of the result of Theorem I.1, we cannot just abandon the data-exchange approach in favor of the rewriting approach when working with valid CQ weakly acyclic materialized-view settings and CQ queries. It is true that we already have a sound and complete rewriting approach to finding all certain answers to CQ queries w.r.t. valid CQ weakly acyclic materialized-view settings under CWA. However, the rewriting approach works via an explicit generate-and-test paradigm for all the candidate certain-answer tuples, please see Note 2 in Section B.3. The advantage of the data-exchange approach in this regard is that we can obtain all the certain-answer tuples for the query Q that are determinable by this (sound) approach, simply by processing Q once on the instance $J_{de}^{\mathcal{M}\Sigma}$, and by then filtering out all the answer tuples that contain null values. In Appendix J we will introduce a sound and complete algorithm for finding all the certain-answer tuples to CQ queries w.r.t. CQ weakly acyclic materialized-view settings under CWA. The algorithm of Appendix J (i) uses the idea and approach of data exchange, and is in fact based on the approach of Section I.1; and (ii) has the same desirable property of “returning all the certain-answer tuples by processing the input query Q once” as just discussed in this paragraph in regard to the data-exchange approach of Section I.1.

(As each of the generate-and-test rewriting algorithm of Section B.3 and the algorithm to be introduced in Appendix J is sound and complete for input instances with CQ queries and CQ weakly acyclic materialized-view settings under CWA, these two approaches have of course the same asymptotic complexity, w.r.t. any relevant complexity measure. Our only argument in the previous paragraph in favor of the algorithm of Appendix J is that that algorithm is, in a sense, more streamlined (than the data-exchange approach), as it does not use the generate-and-test paradigm w.r.t. the candidate certain-answer tuples.)

The reason we are to introduce the algorithm of Appendix J is that, not surprisingly, the data-exchange approach of Section I.1 is not complete under CWA for CQ queries, either for CQ weakly acyclic settings $\mathcal{M}\Sigma$ with $\Sigma = \emptyset$, or for those with $\Sigma \neq \emptyset$. In the remainder of this appendix, we discuss a feature of the data-exchange approach that prevents us from using it as a complete algorithm for this class of input instances under CWA. In Appendix J we will eliminate this feature of the data-exchange approach, in a modification that will give us a sound and complete algorithm for finding all the certain-answer tuples for this class of input instances under CWA.

We now prove that the data-exchange approach is not complete for CQ instances $\mathcal{M}\Sigma$ with $\Sigma = \emptyset$.

EXAMPLE I.1. *We recall the CQ query Q and the CQ views V and W of Example A.1:*

$$\begin{aligned} Q(X, Z) &\leftarrow E(X, Y, Z). \\ V(X, Y) &\leftarrow E(X, Y, Z). \\ W(Y, Z) &\leftarrow E(X, Y, Z). \end{aligned}$$

Using the agreement as in Example B.1 for the constants used in Example A.1, we represent the set of view answers of Example A.1 as $MV = \{ V(c, d), W(d, f) \}$. In the same notation, the tuple \bar{t} of Example A.1 is recast as (c, f) .

Consider the materialized-view setting $\mathcal{M}\Sigma = (\{E\}, \emptyset, \{V, W\}, MV)$, with all the elements as defined above. By definition, $\mathcal{M}\Sigma$ is a CQ weakly acyclic setting. ($\mathcal{M}\Sigma$ is also valid, by the existence of the instance $\{(c, d, f)\}$ of schema $\{E\}$.) The data-exchange approach of Section I.1 applied to $\mathcal{M}\Sigma$ and Q yields the following canonical universal solution, $J_{de}^{\mathcal{M}\Sigma}$, for the source instance MV in the data-exchange setting $\mathcal{S}^{(de)}(\mathcal{M}\Sigma)$:

$$J_{de}^{\mathcal{M}\Sigma} = \{ E(c, d, \perp_1), E(\perp_2, d, f) \}.$$

(The first tuple in $J_{de}^{\mathcal{M}\Sigma}$ is due to the tuple $V(c, d)$ in MV , and the second tuple is due to $W(d, f)$ in MV .) It is easy to see that each of the two answers to the query Q on the instance $J_{de}^{\mathcal{M}\Sigma}$ has nulls and thus cannot qualify as a certain answer to Q w.r.t. $\mathcal{M}\Sigma$. \square

When given as inputs the setting $\mathcal{M}\Sigma$ and query Q of Example I.1, the data-exchange approach of Section I.1 outputs the empty set of candidate-answer tuples. As Q is a CQ query and $\mathcal{M}\Sigma$ is CQ weakly acyclic (with $\Sigma = \emptyset$), the sound and complete rewriting-based algorithm of Section B.3 for finding all the candidate-answer tuples is applicable to $\mathcal{M}\Sigma$ and Q , and outputs $\{(c, f)\}$ when given $\mathcal{M}\Sigma$ and Q in its input. We conclude that the data-exchange approach is incomplete when applied to CQ queries and CQ weakly acyclic settings with $\Sigma = \emptyset$. Further, we can use the example of Appendix E to show that the data-exchange approach is also incomplete when applied to (CQ queries and) CQ weakly acyclic materialized-view settings with $\Sigma \neq \emptyset$.

Why is the data-exchange approach not complete when applied to (CQ queries and) CQ weakly acyclic materialized-view settings? Intuitively, the problem is that its canonical universal solution $J_{de}^{\mathcal{M}\Sigma}$ “may cover too many target instances” (i.e., $J_{de}^{\mathcal{M}\Sigma}$ is an OWA rather than CWA solution). Let us evaluate the queries for the views V and W of Example I.1 over the solution $J_{de}^{\mathcal{M}\Sigma}$ of that example. We obtain that the answer to the view V on $J_{de}^{\mathcal{M}\Sigma}$ is $\{V(c, d), V(\perp_2, d)\}$. Similarly, the answer to W on $J_{de}^{\mathcal{M}\Sigma}$ is $\{W(d, \perp_1), W(d, f)\}$. Thus, if we replace \perp_1 in $J_{de}^{\mathcal{M}\Sigma}$ by any constant except f , or replace \perp_2 by any constant except c , then any ground instance obtained from $J_{de}^{\mathcal{M}\Sigma}$ using these replacements would “generate too many tuples” (as compared with MV) in the answer to either V or W .

We now generalize over this observation. Fix a valid CQ weakly acyclic instance $\mathcal{M}\Sigma$, and consider the canonical universal solution (if one exists) $J_{de}^{\mathcal{M}\Sigma}$ generated by the data-exchange approach with $\mathcal{M}\Sigma$ as input. (In the remainder of this paper, we will refer to $J_{de}^{\mathcal{M}\Sigma}$ as the canonical data-exchange solution for $\mathcal{M}\Sigma$.) By definition of $J_{de}^{\mathcal{M}\Sigma}$, for each $V \in \mathcal{V}$, the answer to the query for V on $J_{de}^{\mathcal{M}\Sigma}$ is a superset of the relation $MV[V]$. Suppose that the answer on $J_{de}^{\mathcal{M}\Sigma}$ to at least one view $V \in \mathcal{V}$ is not a subset of $MV[V]$, as it is the case in the example we have just discussed. Then $J_{de}^{\mathcal{M}\Sigma}$, as a template for instances of schema \mathbf{P} , describes not only

instances that “generate” exactly the set MV in $\mathcal{M}\Sigma$, but also those instances that generate proper supersets of MV . The latter instances are not of interest to us. (Recall that we take the CWA viewpoint, and thus are interested only in the instances I of schema \mathbf{P} such that $\mathcal{V} \Rightarrow_{I, \Sigma} MV$.) As a result, when the data-exchange approach uses $J_{de}^{\mathcal{M}\Sigma}$ to obtain certain answers to the input query Q , it can easily miss those certain answers that characterize only those instances of interest to us.

J. VIEW-VERIFIED DATA EXCHANGE

The problem with the natural data-exchange approach, as introduced in [8, 27], is that its canonical universal solution, when turned into a ground instance, may produce a proper superset of the given set of view answers MV . (See Appendices H–I in this current paper.) That is, the canonical data-exchange solution does not necessarily describe ground solutions for $\mathcal{M}\Sigma$ “tightly enough.” (Recall that we take the CWA viewpoint, and thus are interested only in the instances I of schema \mathbf{P} such that $\mathcal{V} \Rightarrow_{I, \Sigma} MV$. At the same time, the canonical data-exchange solution describes not only these “CWA” instances, but also those that are relevant to the inputs under OWA.)

The approach that we introduce in this appendix builds on data exchange, by “tightening” its universal solutions using $consts(\mathcal{M}\Sigma)$. This approach, which we call *view-verified data exchange*, solves correctly the problem of finding all the candidate-answer tuples w.r.t. a CQ query and a valid CQ weakly acyclic materialized-view setting. We also use the approach of this appendix to solve the problem of deciding whether a given materialized-view setting is valid.

J.1 Chase with MV-Induced Dependencies

In Section J.2 we will define view-verified data exchange for CQ weakly acyclic input instances. (Throughout this appendix, we use the term “CQ weakly acyclic input instance” to refer to a pair $(\mathcal{M}\Sigma, Q)$, where $\mathcal{M}\Sigma$ is a CQ weakly acyclic materialized-view setting, and Q is a CQ query Q over the schema \mathbf{P} in $\mathcal{M}\Sigma$.) Given a $\mathcal{M}\Sigma$ with set of views \mathcal{V} and set of view answers MV , the idea of the approach is to force the canonical data-exchange solution $J_{de}^{\mathcal{M}\Sigma}$ for $\mathcal{M}\Sigma$ to generate only the relations in MV as answers to the queries for \mathcal{V} . (By definition of $J_{de}^{\mathcal{M}\Sigma}$, the answer on $J_{de}^{\mathcal{M}\Sigma}$ to the query for each $V \in \mathcal{V}$ is always a superset of the relation $MV[V]$.) We achieve this goal by chasing $J_{de}^{\mathcal{M}\Sigma}$ using “MV-induced” dependencies. Intuitively, applying MV-induced dependencies to the instance $J_{de}^{\mathcal{M}\Sigma}$ forces some nulls in $J_{de}^{\mathcal{M}\Sigma}$ to become constants in $consts(\mathcal{M}\Sigma)$. As a result of such a chase step, we obtain that for at least one view $V \in \mathcal{V}$, some formerly non-ground tuples in the answer to V on the instance become ground tuples in $MV[V]$.

We now formally define MV-induced dependencies. Let $V(\bar{X}) \leftarrow \phi(\bar{X}, \bar{Y})$ be a CQ query of arity $k_V \geq 0$, and MV be a ground instance of a schema that includes the k_V -ary relation symbol V . First, in case where $MV[V] = \emptyset$, we define the *MV-induced implication constraint* (MV-induced ic) ι_V for V as

$$\iota_V : \phi(\bar{X}, \bar{Y}) \rightarrow \text{false}. \quad (5)$$

(Each MV-induced ic is an implication constraint, i.e.,

a Horn rule with the empty head. See [30] for the discussion and references on implication constraints.)

Second, in case where $k_V \geq 1$, suppose $MV[V] = \{\bar{t}_1, \bar{t}_2, \dots, \bar{t}_{m_V}\}$, with $m_V \geq 1$. Then we define the *MV-induced generalized egd* (*MV-induced ged*) τ_V for V as

$$\tau_V : \phi(\bar{X}, \bar{Y}) \rightarrow \bigvee_{i=1}^{m_V} (\bar{X} = \bar{t}_i). \quad (6)$$

Here, $\bar{X} = [S_1, \dots, S_{k_V}]$ is the head vector of the query for V , with $S_j \in \text{CONST} \cup \text{QVAR}$ for $j \in [1, k_V]$. For each $i \in [1, m_V]$ and for the ground tuple $\bar{t}_i = (c_{i1}, \dots, c_{ik_V}) \in MV[V]$, we abbreviate by $\bar{X} = \bar{t}_i$ the conjunction $\bigwedge_{j=1}^{k_V} (S_j = c_{ij})$. *MV-induced geds* are a straightforward generalization of disjunctive egds of [12, 13].

We now define chase of instances with *MV-induced dependencies*. Consider first *MV-induced implication constraints*. Given an instance K of schema \mathbf{P} and an *MV-induced ic* ι_V as in Eq. (5), suppose there exists a homomorphism h from the antecedent $\phi(\bar{X}, \bar{Y})$ of ι_V to K . The intuition here is that we want to make sure that K does not “generate” any tuples in the relation $MV[V]$; however, by the existence of h , the instance K does generate at least one such tuple. We then say that *chase with ι_V (and h) fails on the instance K and produces the set $\{\epsilon\}$* , with ϵ denoting the empty instance.

Now let τ as in Eq. (6) be an *MV-induced generalized egd* for a $V \in \mathcal{V}$. The intuition here is that K must “generate” *only* the tuples in the relation $MV[V]$; we make this happen by assigning nulls in K to constants in $MV[V]$. (If such assignments are not possible, chase with τ fails on K .) Example J.1 is the running example.

Our definition of the chase step with τ as in Eq. (6) is a straightforward extension of the definition of [13] for their disjunctive egds, as follows. Consider the consequent of τ , of the form $\bigvee_{i=1}^{m_V} (\bar{X} = \bar{t}_i)$. Recall that for each $i \in [1, m_V]$, the expression $\bar{X} = \bar{t}_i$ is of the form $\bigwedge_{j=1}^{k_V} (S_j = c_{ij})$. Denote by $\tau^{(1)}, \dots, \tau^{(m_V)}$ the following m_V dependencies obtained from τ : $(\phi(\bar{X}, \bar{Y}) \rightarrow \bar{X} = \bar{t}_1), \dots, (\phi(\bar{X}, \bar{Y}) \rightarrow \bar{X} = \bar{t}_{m_V})$, and call them the *dependencies associated with τ* . For each $i \in [1, m_V]$, $\tau^{(i)}$ is an embedded dependency that can be equivalently represented by k_V egds $\tau^{(i,1)}, \dots, \tau^{(i,k_V)}$. Here, for each $j \in [1, k_V]$, the egd $\tau^{(i,j)}$ is $\phi(\bar{X}, \bar{Y}) \rightarrow S_j = c_{ij}$.

Given a τ as in Eq. (6) and an instance K of schema \mathbf{P} , suppose that there exists a homomorphism h from $\phi(\bar{X}, \bar{Y})$ to K such that $\bigwedge_{j=1}^{k_V} (h(S_j) = h(c_{ij}))$ is not a tautology for any $i \in [1, m_V]$. Then we say that τ is *applicable to K with the homomorphism h* . It is easy to see that it is also the case that each of $\tau^{(1)}, \dots, \tau^{(m_V)}$ can be applied to K with h . That is, for each $i \in [1, m_V]$, the chase of K is applicable with at least one egd $\tau^{(i,j)}$ in the equivalent representation of $\tau^{(i)}$ as a set of egds. For each $i \in [1, m_V]$, let K_i be the result of applying all the egds $\tau^{(i,1)}, \dots, \tau^{(i,k_V)}$ to K with h . Note that chase with $\tau^{(i,j)}$ and h can fail on K for some i and j . For each such i , we say that *chase with $\tau^{(i)}$ fails on K and produces the empty instance ϵ* .

Similarly to [13], we distinguish two cases:

- If the set $\{K_1, \dots, K_{m_V}\}$ contains only empty instances, we say that *chase with τ (and h) fails on K and produces the set $\{\epsilon\}$* .

- Otherwise, let $\mathcal{K}^{(\tau)} = \{K_{i_1}, \dots, K_{i_p}\}$ be the set of all nonempty elements of $\{K_1, \dots, K_{m_V}\}$. We say that $\mathcal{K}^{(\tau)}$ is the *result of applying τ to K with h* .

Similarly to the approach of [13], in addition to chase steps with *MV-induced dependencies* we will also use chase steps with egds and tgds as in Section 3.2. For the chase step of each type, we will use the set notation for uniformity: $K \Rightarrow^{\sigma, h} K'$ denotes that a chase step with dependency σ and homomorphism h applied to instance K yields a set of instances K' . Whenever chase with an egd fails on K , the set K' is the set $\{\epsilon\}$ by convention; in all other cases where σ is an egd or a tgd, the set K' is a singleton set. For σ of the form as in Eq. (5)–(6), the set K' is in some cases $\{\epsilon\}$ as defined above.

DEFINITION J.1. (*MV-enhanced chase*) Let Σ be a set of egds and tgds, let $\Sigma^{(MV)}$ be a set of *MV-induced dependencies*, and let K be an instance.

- A chase tree of K with $\Sigma \cup \Sigma^{(MV)}$ is a tree (finite or infinite) such that:
 - The root is K , and
 - For every node K_j in the tree, let K_j be the set of its children. Then there must exist some dependency σ in $\Sigma \cup \Sigma^{(MV)}$ and some homomorphism h such that $K_j \Rightarrow^{\sigma, h} K_j$.
- A finite *MV-enhanced chase* of K with $\Sigma \cup \Sigma^{(MV)}$ is a finite chase tree \mathcal{T} , such that for each leaf K_p of \mathcal{T} , we have that either (a) K_p is ϵ , or (b) there is no dependency σ in $\Sigma \cup \Sigma^{(MV)}$ and no homomorphism h such that σ can be applied to K_p with h .

□

EXAMPLE J.1. Consider Q as in Example A.1 and $\mathcal{M}\Sigma = (\{E\}, \emptyset, \{V, W\}, MV)$, with all the elements except MV as in Example A.1.⁸ For this current example, we define the set MV as

$$MV = \{ V(c, d), V(g, d), W(d, f) \}.$$

By definition, $\mathcal{M}\Sigma$ paired with Q is a CQ instance with $\Sigma = \emptyset$. $\mathcal{M}\Sigma$ is also valid, as witnessed by the instance $\{E(c, d, f), E(g, d, f)\}$. The data-exchange approach of Appendix H yields the following canonical data-exchange solution $J_{de}^{\mathcal{M}\Sigma}$ for $\mathcal{M}\Sigma$:

$$J_{de}^{\mathcal{M}\Sigma} = \{ E(c, d, \perp_1), E(g, d, \perp_2), E(\perp_3, d, f) \}.$$

The set of answers without nulls to the query Q on $J_{de}^{\mathcal{M}\Sigma}$ is empty. Thus, the data-exchange approach applied to $\mathcal{M}\Sigma$ discovers no certain answers to the query Q w.r.t. the setting $\mathcal{M}\Sigma$.

In applying the view-verified data-exchange approach to the input $(\mathcal{M}\Sigma, Q)$, we first construct the *MV-induced generalized egds*, τ_V and τ_W , one for each of the two views in $\mathcal{M}\Sigma$. (As MV has no empty relations, we do not need to construct *MV-induced ics* for $\mathcal{M}\Sigma$.)

$$\begin{aligned} \tau_V &: E(X, Y, Z) \rightarrow (X = c \wedge Y = d) \vee (X = g \wedge Y = d). \\ \tau_W &: E(X, Y, Z) \rightarrow (Y = d \wedge Z = f). \end{aligned}$$

⁸Please see Example I.1 for the details.

The two dependencies associated with τ_V are $\tau_V^{(1)} : E(X, Y, Z) \rightarrow (X = c \wedge Y = d)$ and $\tau_V^{(2)} : E(X, Y, Z) \rightarrow (X = g \wedge Y = d)$. Each of $\tau_V^{(1)}$ and $\tau_V^{(2)}$ can be equivalently represented by two egds. For instance, the egd representation for $\tau_V^{(1)}$ is via $\tau_V^{(1,1)} : E(X, Y, Z) \rightarrow X = c$ and $\tau_V^{(1,2)} : E(X, Y, Z) \rightarrow Y = d$. Similarly, there is one dependency $\tau_W^{(1)} (= \tau_W)$ associated with τ_W ; an equivalent representation of $\tau_W^{(1)}$ is via two egds.

Consider a homomorphism $h_V^{(1)} : \{X \rightarrow c, Y \rightarrow d, Z \rightarrow \perp_1\}$ from the antecedent $E(X, Y, Z)$ of τ_V to the instance $J_{de}^{\mathcal{M}\Sigma}$. As applying $h_V^{(1)}$ to the consequent of $\tau_V^{(1)}$ gives us the tautology $(c = c \wedge d = d)$, we conclude that τ_V is not applicable to $J_{de}^{\mathcal{M}\Sigma}$ with $h_V^{(1)}$.

Consider now the homomorphism $h_V^{(2)} : \{X \rightarrow \perp_3, Y \rightarrow d, Z \rightarrow f\}$ from the antecedent of τ_V to $J_{de}^{\mathcal{M}\Sigma}$. Applying $h_V^{(2)}$ to the consequent of τ_V gives us the expression $(\perp_3 = c \wedge d = d) \vee (\perp_3 = g \wedge d = d)$, which has no tautologies among its disjuncts. Thus, τ_V is applicable to $J_{de}^{\mathcal{M}\Sigma}$ with $h_V^{(2)}$. The chase step with τ_V and $h_V^{(2)}$ transforms $J_{de}^{\mathcal{M}\Sigma}$ into instances J_1 and J_2 , as follows.

$$J_1 = \{ E(c, d, \perp_1), E(g, d, \perp_2), E(c, d, f) \}.$$

$$J_2 = \{ E(c, d, \perp_1), E(g, d, \perp_2), E(g, d, f) \}.$$

(J_1 results from assigning $\perp_3 := c$, and J_2 from $\perp_3 := g$.)

We then use the same procedure to apply τ_W to each of J_1 and J_2 . In each case, the chase steps assign the value f to each of \perp_1 and \perp_2 . As a result, the following instance $J_{vv}^{\mathcal{M}\Sigma}$ is obtained from each of J_1 and J_2 :

$$J_{vv}^{\mathcal{M}\Sigma} = \{ E(c, d, f), E(g, d, f) \}.$$

□

J.2 Solving CQ Weakly Acyclic Instances

We now define the view-verified data-exchange approach to the problem of finding all certain answers to queries w.r.t. materialized-view settings.

Let $\mathcal{M}\Sigma = (\mathbf{P}, \Sigma, \mathcal{V}, MV)$ be a CQ materialized-view setting. Then the set $\Sigma^{(\mathcal{M}\Sigma)}$ of MV-induced dependencies for $\mathcal{M}\Sigma$ is a set of up to $|\mathcal{V}|$ elements, as follows. For each $V \in \mathcal{V}$ such that $k_V \neq 0$ or $MV[V] \neq \{\emptyset\}$, $\Sigma^{(\mathcal{M}\Sigma)}$ has one MV-induced implication constraint or one MV-induced generalized egd, by the rules as in Eq. (5)–(6) in Section J.1.⁹

For CQ weakly acyclic input instances $(\mathcal{M}\Sigma, Q)$ we introduce the following *view-verified data-exchange approach to finding certain query answers w.r.t. a materialized-view setting*. First, we compute (as in Appendix H) the canonical universal solution $J_{de}^{\mathcal{M}\Sigma}$ for the source instance MV in the data-exchange setting $\mathcal{S}^{(de)}(\mathcal{M}\Sigma)$. If $J_{de}^{\mathcal{M}\Sigma}$ does not exist, we stop and output the answer that $\mathcal{M}\Sigma$ is not valid. Otherwise we obtain a chase tree of $J_{de}^{\mathcal{M}\Sigma}$ with $\Sigma \cup \Sigma^{(\mathcal{M}\Sigma)}$, where $\Sigma^{(\mathcal{M}\Sigma)}$ is the set of

⁹We omit from $\Sigma^{(\mathcal{M}\Sigma)}$ the dependencies, of the form $\phi(\bar{X}, \bar{Y}) \rightarrow \text{true}$, for the case where $k_V = 0$ and $MV[V] \neq \emptyset$. By the results in this appendix, adding these dependencies to $\Sigma^{(\mathcal{M}\Sigma)}$ would not change any chase results.

MV-induced dependencies for $\mathcal{M}\Sigma$. If the chase tree is finite, denote by $\mathcal{J}_{vv}^{\mathcal{M}\Sigma}$ the set of all the nonempty leaves of the tree. We call each $J \in \mathcal{J}_{vv}^{\mathcal{M}\Sigma}$ a *view-verified universal solution* for $\mathcal{M}\Sigma$. If $\mathcal{J}_{vv}^{\mathcal{M}\Sigma} = \emptyset$, then we stop and output the answer that $\mathcal{M}\Sigma$ is not valid. Otherwise, for each $J \in \mathcal{J}_{vv}^{\mathcal{M}\Sigma}$ we compute the set $Q_{\downarrow}(J)$ of all the tuples in $Q(J)$ that do not contain nulls. Finally, the output of the approach for the input $(\mathcal{M}\Sigma, Q)$ is the set

$$\bigcap_{J \in \mathcal{J}_{vv}^{\mathcal{M}\Sigma}} Q_{\downarrow}(J). \quad (7)$$

The view-verified data-exchange approach to the problem of finding all certain answers to queries w.r.t. materialized-view settings addresses the shortcoming of the data-exchange approach, see Appendix H. Recall that the canonical universal solution $J_{de}^{\mathcal{M}\Sigma}$ of the latter approach might not cover “tightly enough” all the instances of interest to the attackers. In the view-verified approach, we address this problem, by using our extension of the chase to generate from $J_{de}^{\mathcal{M}\Sigma}$ a set $\mathcal{J}_{vv}^{\mathcal{M}\Sigma}$ of instances that are each “tighter” than $J_{de}^{\mathcal{M}\Sigma}$ in this sense.

In Section J.3 we will show that the view-verified data-exchange approach is a sound and complete algorithm for the problem of finding all certain answers to queries w.r.t. materialized-view settings, in all cases where the input instances are CQ weakly acyclic. In particular, we will see that the set $\mathcal{J}_{vv}^{\mathcal{M}\Sigma}$ is well defined, in that the chase tree in the view-verified data-exchange approach is always finite. We will also see that the set $\mathcal{J}_{vv}^{\mathcal{M}\Sigma}$ is “just tight enough,” in the following sense: Recall (see Section 4.1) the definition of *certain _{$\mathcal{M}\Sigma$} (Q)*, i.e., of the set of certain answers of query Q w.r.t. materialized-view setting $\mathcal{M}\Sigma$. Then the expression in Eq. (7), which is the intersection of all the “certain-answer expressions” for Q and for the individual elements of the set $\mathcal{J}_{vv}^{\mathcal{M}\Sigma}$, is exactly the set *certain _{$\mathcal{M}\Sigma$} (Q)*.

EXAMPLE J.2. Recall the input instance $(\mathcal{M}\Sigma, Q)$ of Example J.1, and the instance $J_{vv}^{\mathcal{M}\Sigma}$ obtained in that example. $J_{vv}^{\mathcal{M}\Sigma}$ is the (only) view-verified universal solution for $\mathcal{M}\Sigma$. The set of answers without nulls to the query Q on $J_{vv}^{\mathcal{M}\Sigma}$ is $\{(c, f), (g, f)\}$. Thus, (c, f) and (g, f) are certain answers of the query Q w.r.t. the materialized-view setting $\mathcal{M}\Sigma$, as computed for the instance $(\mathcal{M}\Sigma, Q)$ by the view-verified data-exchange approach. Both (c, f) and (g, f) (and nothing else) are also discovered by the rewriting algorithm of Appendix B, which is sound and complete for $(\mathcal{M}\Sigma, Q)$. (See Note 2 in Section B.3.) □

J.3 Correctness, Validity, and Complexity

In this subsection, we show that the view-verified data-exchange approach is sound and complete for all CQ weakly acyclic input instances, and discuss its runtime and space complexity. We also show how the approach can be used to decide whether a CQ weakly acyclic materialized-view setting $\mathcal{M}\Sigma$ is valid.

View-verified data exchange is an algorithm. We begin by obtaining a basic observation that builds on the results of [13] for chase with tgds and disjunctive egds (as they are defined in [13]). It is immediate from Proposition J.1 that view-verified data exchange always terminates in finite time for CQ weakly acyclic inputs.

PROPOSITION J.1. *Given a CQ weakly acyclic materialized-view setting $\mathcal{M}\Sigma$, such that its canonical data-exchange solution $J_{de}^{\mathcal{M}\Sigma}$ exists. Assume that everything in $\mathcal{M}\Sigma$ is fixed except for the instance MV . Then we have that:*

- (1) *MV-enhanced chase of $J_{de}^{\mathcal{M}\Sigma}$ with $\Sigma \cup \Sigma^{(\mathcal{M}\Sigma)}$ is a finite tree, \mathcal{T} , such that:*
 - (a) *\mathcal{T} is of polynomial depth in the size of MV , and*
 - (b) *The number of leaves in \mathcal{T} is up to exponential in the size of MV ; and*
- (2) *For each nonempty leaf J of \mathcal{T} , we have that:*
 - (a) *J is of polynomial size in the size of MV , and*
 - (b) *Each grounded version of J is a Σ -valid base instance for \mathcal{V} and MV .*

□

(A grounded version of instance K results from replacing consistently all its nulls with distinct new constants.)

The proof of Proposition J.1 relies heavily on the results of [13], particularly on its Theorem 3.9. Recall the “decomposition,” in Section J.1, of MV -induced generalized egds into egds that are defined as in Section 3.2. Intuitively, given a CQ weakly acyclic materialized-view setting $\mathcal{M}\Sigma$ and for each node K on each path from the root $J_{de}^{\mathcal{M}\Sigma}$ of the tree \mathcal{T} for $\mathcal{M}\Sigma$, we can obtain K by chasing the root of \mathcal{T} using only egds and weakly acyclic tgds.¹⁰ The key observation here is that even though the set $\Sigma^{(\mathcal{M}\Sigma)}$ of dependencies is not fixed (in fact, its size is linear in the size of the instance MV in $\mathcal{M}\Sigma$), all the constants that contribute to the size of $\Sigma^{(\mathcal{M}\Sigma)}$ are already used in the root $J_{de}^{\mathcal{M}\Sigma}$ of the tree \mathcal{T} , by definition of $J_{de}^{\mathcal{M}\Sigma}$. In addition, the antecedent of each MV -induced generalized egd in $\Sigma^{(\mathcal{M}\Sigma)}$ is of constant size, by definition of the size of $\mathcal{M}\Sigma$. As a result, we can build on Theorem 3.9 and Proposition 5.6 of [13] to obtain items (1)(a) and (2)(a) of our Proposition J.1.

Item (2)(b) of Proposition J.1 is by definition of MV -enhanced chase, and (1)(b) is by construction of the tree \mathcal{T} . Appendix K provides a lower bound, via an example where for a CQ instance $\mathcal{M}\Sigma$ with $\Sigma = \emptyset$, the number of leaves in a chase tree is exponential in the size of $\mathcal{M}\Sigma$.

Soundness and completeness. By Proposition J.1 (2)(b), the view-verified data-exchange approach is a *complete* algorithm when applied to CQ weakly acyclic input instances $(\mathcal{M}\Sigma, Q)$. (That is, for each certain-answer tuple \bar{t} for a problem input in this class, view-verified data exchange outputs \bar{t} .) We now make a key observation toward a proof that this algorithm is also *sound* for such instances. (Soundness means that for each tuple \bar{t} that this approach outputs for an input $(\mathcal{M}\Sigma, Q)$ in this class, \bar{t} is a certain-answer tuple for $(\mathcal{M}\Sigma, Q)$.)

PROPOSITION J.2. *Given a CQ weakly acyclic materialized-view setting $\mathcal{M}\Sigma = (\mathbf{P}, \Sigma, \mathcal{V}, MV)$ and a CQ query Q . Then, for each instance I such that $\mathcal{V} \Rightarrow_{I, \Sigma} MV$, there exists a homomorphism from some view-verified universal solution for $\mathcal{M}\Sigma$ to I .* □

¹⁰Besides the egds and tgds of Section 3.2, chase on each path in \mathcal{T} may use MV -induced implication constraints. However, the only role of the latter constraints is to obtain the instance ϵ and thus to terminate the respective path in \mathcal{T} .

The intuition for the proof of Proposition J.2 is as follows. For a given $\mathcal{M}\Sigma$, whenever an instance I exists such that $\mathcal{V} \Rightarrow_{I, \Sigma} MV$, a canonical data-exchange solution $J_{de}^{\mathcal{M}\Sigma}$ for $\mathcal{M}\Sigma$ must also exist. By definition of $J_{de}^{\mathcal{M}\Sigma}$, there must be a homomorphism from $J_{de}^{\mathcal{M}\Sigma}$ to the instance I . We then start applying MV -enhanced chase to $J_{de}^{\mathcal{M}\Sigma}$, to simulate some rooted path, $P_{(\mathcal{T})}$, in the chase tree \mathcal{T} for $\mathcal{M}\Sigma$. (The tree is finite by Proposition J.1.) In following the path $P_{(\mathcal{T})}$ via the chase, we make sure that there is a homomorphism from each node in the path to I , by always choosing an “appropriate” associated dependency $\tau^{(i)}$ for each MV -induced generalized egd τ that we are applying in the chase. By $\mathcal{V} \Rightarrow_{I, \Sigma} MV$, such a choice always exists, and the path $P_{(\mathcal{T})}$ terminates in finite time in a nonempty instance, J . By definition, J is a view-verified universal solution for $\mathcal{M}\Sigma$. By our simulation of the path $P_{(\mathcal{T})}$ “on the way to” I , there exists a homomorphism from J to I .

Validity of setting $\mathcal{M}\Sigma$. By the results of [13], when for a given $\mathcal{M}\Sigma$ no canonical data-exchange solution exists, then $\mathcal{M}\Sigma$ is not a valid setting. We refine this observation into a sufficient and necessary condition for validity of CQ weakly acyclic materialized-view settings $\mathcal{M}\Sigma$. (The only-if part of Proposition J.3 follows from Proposition J.2, and its if part is by Proposition J.1 (2)(b).)

PROPOSITION J.3. *Given a CQ weakly acyclic materialized-view setting $\mathcal{M}\Sigma$, the setting $\mathcal{M}\Sigma$ is valid iff the set $\mathcal{J}_{vv}^{\mathcal{M}\Sigma}$ of view-verified universal solutions for $\mathcal{M}\Sigma$ is not empty.* □

Correctness of view-verified data exchange. By Proposition J.2, view-verified data exchange is sound. By Proposition J.3, it outputs a set of certain-answer tuples iff its input is valid. We now conclude:

THEOREM J.1. *View-verified data exchange is a sound and complete algorithm for finding all certain answers to CQ queries w.r.t. CQ weakly acyclic materialized-view-settings.* □

Complexity of view-verified data exchange for CQ weakly acyclic input instances. By Theorem J.1, view-verified data exchange is an algorithm for all CQ weakly acyclic input instances. We now obtain an exponential-time upper bound on the runtime complexity of the view-verified data-exchange approach, as follows.

Given a CQ weakly acyclic input instance $(\mathcal{M}\Sigma, Q)$, the runtime of the approach of Section J.2 is exponential in the size of Q and of the set of answers MV in $\mathcal{M}\Sigma$, assuming that the rest of $\mathcal{M}\Sigma$ is fixed. This complexity setting extends naturally that of [31]: Zhang and Mendelzon in [31] assumed for their problem that the base schema and the view definitions are fixed, whereas the set of view answers MV and the queries posed on the base schema in presence of MV can vary. The authors of [31] did not consider dependencies on the base schema; we follow the standard data-exchange assumption, see, e.g., [13], that the given dependencies are fixed rather than being part of the problem input.

To obtain the above exponential-time upper bound for the problem of view-verified data exchange for CQ weakly acyclic input instances $(\mathcal{M}\Sigma, Q)$, we analyze the following flow for the view-verified data-exchange algorithm of Section J.2. First, we spend exponential time

in the arity k of Q to generate all the k -ary ground tuples \bar{t} out of the set $\text{consts}(\mathcal{M}\Sigma)$. (Generating each such \bar{t} gives rise to one iteration of the *main loop* of the algorithm.) For each such tuple \bar{t} , we then do the following:

- Construct the query $Q(\bar{t})$, as the result of applying to the query Q the homomorphism¹¹ μ , such that (i) μ maps the head vector \bar{X} of Q to \bar{t} , and (ii) μ is the identity mapping on each term that occurs in Q but not in its head vector \bar{X} ;
- Enumerate all the (up to an exponential number of) view-verified universal solutions J for $\mathcal{M}\Sigma$ (recall that generating each such J takes polynomial time in the size of MV , see Proposition J.1); and then
- For each such J that is not the empty instance, verify whether the query $Q(\bar{t})$ has a nonempty set of answers, which would be precisely $\{\bar{t}\}$, on the instance J . (For each \bar{t} generated as above, we use a one-bit flag to track whether \bar{t} is an answer to Q on all such instances J ; each \bar{t} that is an answer to Q on all the instances J is returned as an answer tuple by the view-verified data-exchange algorithm.) The runtime for this verification step is polynomial in the size of MV (because the size of J is polynomial in the size of MV , see Proposition J.1) and is exponential in the number of subgoals of Q . (As the schema \mathbf{P} in $\mathcal{M}\Sigma$ is fixed, each subgoal of the query Q has up to constant arity.)

Observe that for each tuple \bar{t} generated in the main loop of the algorithm, the respective iteration of the main loop runs in PSPACE. Indeed, recall from Proposition J.1 that each instance J as above is of size polynomial in the size of the instance MV in $\mathcal{M}\Sigma$. Further, the size of each candidate valuation from $Q(\bar{t})$ to J is linear in the size of Q ; thus, we satisfy the PSPACE requirement as long as we generate these candidate valuations one at a time (“on the fly” for each fixed J), in some clear algorithmic order.

Further, the entire view-verified data-exchange algorithm (i.e., finding *all* the certain-answer tuples for the given input CQ weakly acyclic pair $(\mathcal{M}\Sigma, Q)$) also runs in PSPACE, provided that we:

- Output each certain-answer tuple \bar{t} “on the fly” (i.e., as soon as we know that it is a certain answer), and
- Use a counter (e.g., a binary-number representation of each k -ary ground candidate certain-answer tuple \bar{t} , as generated in the main loop of the algorithm) to keep track of the “latest” \bar{t} that we have looked at and to generate from that “latest” \bar{t} the next candidate certain-answer tuple \bar{t} that we are to examine for the given input; the size of such a counter would be polynomial in the size of the problem input.

K. THE NUMBER OF LEAVES IN MV -ENHANCED CHASE CAN BE EXPONENTIAL IN THE SIZE OF THE INPUT

¹¹It is easy to verify that if a homomorphism μ specified by (i)-(ii) does not exist, then \bar{t} cannot be a certain answer to Q w.r.t. $\mathcal{M}\Sigma$.

In this appendix we show by example a family of CQ materialized-view settings $\mathcal{M}\Sigma$ with $\Sigma = \emptyset$, such that the number of leaves in a chase tree for each setting in the family is exponential in the size of the setting. As usual and similarly to [31], we assume that the *size* of a given materialized-view setting $\mathcal{M}\Sigma$ is the size of its instance MV , with the remaining elements of $\mathcal{M}\Sigma$ being fixed. (See Section J.3 for a detailed discussion.)

EXAMPLE K.1. Consider a schema \mathbf{P} with two binary relations P and R , and with $\Sigma = \emptyset$. Let the set of views $\mathcal{V} = \{V, W\}$ be defined via two CQ queries, as follows:

$$\begin{aligned} V(X) &\leftarrow P(X, Y), R(Y, Z). \\ W(Z) &\leftarrow R(Y, Z). \end{aligned}$$

For each $n \geq 1$, consider a set $MV_{(n)}$ of answers for \mathcal{V} , with $n + 2$ tuples, as follows. The relation $MV_{(n)}[V]$ has n tuples $V(1), V(2), \dots, V(n)$, and $MV_{(n)}[W]$ has tuples $W(0)$ and $W(1)$.

For each $n \geq 1$, let the materialized-view setting $\mathcal{M}\Sigma^{(n)}$ be the tuple $(\mathbf{P}, \Sigma, \mathcal{V}, MV_{(n)})$, with all the components as described above. (As specified above, the set Σ is the empty set for each $n \geq 1$.)

The canonical universal solution $J_{de}^{\mathcal{M}\Sigma^{(n)}}$ for $MV_{(n)}$ has two tuples, $P(i, \perp_{(i,1)})$ and $R(\perp_{(i,1)}, \perp_{(i,2)})$, for $V(i)$ in $MV_{(n)}$, for each $i \in [1, n]$. It also has the tuples $R(\perp_{(n+1,1)}, 0)$ and $R(\perp_{(n+2,1)}, 1)$ for $MV_{(n)}[W]$.

The process of creating view-verified universal solutions for $\mathcal{M}\Sigma^{(n)}$ involves assigning either 0 or 1 independently to each of the nulls $\perp_{(i,2)}$, for all $i \in [1, n]$. It is easy to see that this process creates 2^n nonisomorphic instances, one for each assignment of zeroes and ones to each element of the vector $[\perp_{(1,2)}, \perp_{(2,2)}, \dots, \perp_{(n,2)}]$. The expression 2^n is exponential in the size of the set $MV_{(n)}$ of view answers in $\mathcal{M}\Sigma^{(n)}$. \square

L. CHASE CANNOT BE STAGED FOR FINDING ALL CERTAIN ANSWERS

In this appendix we provide two examples that show that in the problem of finding all certain answers to a CQ query w.r.t. a CQ weakly acyclic materialized-view setting, one cannot always find all the certain answers correctly if one does the chase (in view-verified data-exchange, see Appendix J) *in stages*. That is, chase only with the input dependencies Σ , followed by chase only with the “ MV -induced dependencies,” does not always yield a correct solution. (This is the point of Example L.1.) The reverse order of the “stages” does not always work either. (This is the point of Example L.2.)

EXAMPLE L.1. Consider a schema $\mathbf{P} = \{P, S\}$ with binary relation symbols P and S . Let σ be a dependency defined on the schema \mathbf{P} , as follows. (The dependency σ is an egd, specifically a functional dependency.)

$$\sigma : P(X, Y) \wedge P(X, Z) \rightarrow Y = Z.$$

Further, let U, V , and W be three CQ views over \mathbf{P} , and let MV be the set of answers for these views, as follows.

$U(X) \leftarrow P(X, Y), S(Y, Z).$
 $V(X) \leftarrow P(X, Y).$
 $W(X, Z) \leftarrow S(X, Y), P(Y, Z).$
 $MV = \{U(c), V(c), W(g, h)\}.$

We denote the set $\{U, V, W\}$ by \mathcal{V} , and the set $\{\sigma\}$ by Σ . Then the setting $\mathcal{M}\Sigma = (\mathbf{P}, \Sigma, \mathcal{V}, MV)$ is CQ weakly acyclic.

Now let Q be a CQ query:

$Q(X) \leftarrow S(X, Y).$

We consider the problem of finding the set of certain answers to the query Q w.r.t. the setting $\mathcal{M}\Sigma$ using the view-verified data-exchange approach, as described in Appendix J. By this approach, we first construct, from the materialized-view setting $\mathcal{M}\Sigma$, a data-exchange setting $\mathcal{S}^{(de)}(\mathcal{M}\Sigma) = (\mathcal{V}, \mathbf{P}, \Sigma_{st} \cup \Sigma)$. Here, $\Sigma_{st} = \{\sigma_U, \sigma_V, \sigma_W\}$ is the set of the following three tgds:

$\sigma_U : U(X) \rightarrow \exists Y, Z \quad P(X, Y) \wedge S(Y, Z).$
 $\sigma_V : V(X) \rightarrow \exists Y \quad P(X, Y).$
 $\sigma_W : W(X, Z) \rightarrow \exists Y \quad S(X, Y) \wedge P(Y, Z).$

We then designate the set of view answers MV in the materialized-view setting $\mathcal{M}\Sigma$ to be a source instance for the data-exchange setting $\mathcal{S}^{(de)}(\mathcal{M}\Sigma)$.

We now proceed to construct the canonical universal solution, call it J_0 , for the source instance MV in the data-exchange setting $\mathcal{S}^{(de)}(\mathcal{M}\Sigma)$:

$J_0 = \{P(c, \perp_1), S(\perp_1, \perp_2), P(c, \perp_3),$
 $S(g, \perp_4), P(\perp_4, h)\}.$

In the instance J_0 , the atoms $P(c, \perp_1)$ and $S(\perp_1, \perp_2)$ are due to the atom $U(c)$ in MV and to the tgd σ_U , and so on for the rest of MV and of Σ_{st} .

As described in Appendix J, toward finding all the certain answers to the query Q w.r.t. the materialized-view setting $\mathcal{M}\Sigma$, we now chase the instance J_0 , using both the set $\Sigma = \{\sigma\}$ in $\mathcal{M}\Sigma$, as well as the dependencies $\tau_{U(c)}$, $\tau_{V(c)}$, and $\tau_{W(g, h)}$, as follows. (The three latter dependencies are generated by the view-verified data-exchange approach from the inputs \mathcal{V} and MV .)

$\tau_{U(c)} : P(X, Y) \wedge S(Y, Z) \rightarrow X = c.$
 $\tau_{V(c)} : P(X, Y) \rightarrow X = c.$
 $\tau_{W(g, h)} : S(X, Y) \wedge P(Y, Z) \rightarrow X = g \wedge Z = h.$

We do three stages of the chase of the instance J_0 with σ , $\tau_{U(c)}$, $\tau_{V(c)}$, and $\tau_{W(g, h)}$. In Stages I and III, we perform the chase steps with the input dependency σ on the schema \mathbf{P} , and in Stage II, we chase the instance with the MV-induced dependencies $\tau_{U(c)}$, $\tau_{V(c)}$, and $\tau_{W(g, h)}$.

Stage I: A chase step of the instance J_0 with the egd σ turns the atom $P(c, \perp_3)$ of J_0 into a copy of its atom $P(c, \perp_1)$, resulting in the following instance J_1 (in which we drop the duplicate of $P(c, \perp_1)$):

$J_1 = \{P(c, \perp_1), S(\perp_1, \perp_2), S(g, \perp_4), P(\perp_4, h)\}.$

The egd σ does not apply to the instance J_1 .

Stage II: We now chase the instance J_1 with the MV-induced dependencies $\tau_{U(c)}$, $\tau_{V(c)}$, and $\tau_{W(g, h)}$. The egd $\tau_{V(c)}$ applies to the atom $P(\perp_4, h)$ in J_1 , turning it into $P(c, h)$ and, as a side effect, also turning the atom $S(g, \perp_4)$ of J_1 into $S(g, c)$. We call the resulting instance J_2 :

$J_2 = \{P(c, \perp_1), S(\perp_1, \perp_2), S(g, c), P(c, h)\}.$

The MV-induced dependencies $\tau_{U(c)}$, $\tau_{V(c)}$, and $\tau_{W(g, h)}$ do not apply to the instance J_2 .

Note that if we stop after this Stage II, the set of answers without nulls to the query Q on the instance J_2 is $Q(J_2) = \{(g)\}$. However, we observe that the instance J_2 does not satisfy the egd σ . We can then do Stage III of the chase, by applying σ to the instance J_2 . The application binds the null \perp_1 to the constant h , in the atoms $P(c, \perp_1)$ and $S(\perp_1, \perp_2)$ of the instance J_2 . We call the resulting instance J_3 :

$J_3 = \{P(c, h), S(h, \perp_2), S(g, c)\}.$

The set of answers without nulls to the query Q on the instance J_3 is $Q(J_3) = \{(g), (h)\}$. By the results reported in Appendix J, this set is a correct set of certain answers to Q w.r.t. the given materialized-view setting $\mathcal{M}\Sigma$. We can see that by not applying Stage III in the chase, we would have missed the certain answer (h) to the query Q w.r.t. the setting $\mathcal{M}\Sigma$. \square

EXAMPLE L.2. Consider a schema $\mathbf{P} = \{P, S\}$ with a unary relation symbol P and a binary relation symbol S . Let σ be a dependency (specifically, a full tgd) defined on the schema \mathbf{P} , as follows.

$\sigma : S(X, Y) \rightarrow P(Y).$

Further, let U and V be two CQ views over \mathbf{P} , and let MV be the set of answers for these views, as follows.

$U(X) \leftarrow P(X), S(X, Y).$
 $V(X) \leftarrow P(X).$
 $MV = \{U(c), V(c)\}.$

We denote the set $\{U, V\}$ by \mathcal{V} , and the set $\{\sigma\}$ by Σ . Then the setting $\mathcal{M}\Sigma = (\mathbf{P}, \Sigma, \mathcal{V}, MV)$ is CQ weakly acyclic.

Now let Q be a CQ query:

$Q(X) \leftarrow S(X, X).$

We consider the problem of finding the set of certain answers to the query Q w.r.t. the setting $\mathcal{M}\Sigma$, using the view-verified data-exchange approach detailed in Appendix J. By this approach, we first construct, from the materialized-view setting $\mathcal{M}\Sigma$, a data-exchange setting $\mathcal{S}^{(de)}(\mathcal{M}\Sigma) = (\mathcal{V}, \mathbf{P}, \Sigma_{st} \cup \Sigma)$. Here, $\Sigma_{st} = \{\sigma_U, \sigma_V\}$ is the set of the following two tgds:

$\sigma_U : U(X) \rightarrow \exists Y \quad P(X) \wedge S(X, Y).$
 $\sigma_V : V(X) \rightarrow P(X).$

We then designate the set of view answers MV in the materialized-view setting $\mathcal{M}\Sigma$ to be a source instance for the data-exchange setting $\mathcal{S}^{(de)}(\mathcal{M}\Sigma)$.

We proceed to construct the canonical universal solution, call it J_0 , for the source instance MV in the data-exchange setting $\mathcal{S}^{(de)}(\mathcal{M}\Sigma)$.

$J_0 = \{P(c), S(c, \perp_1)\}.$

In the instance J_0 , the atoms $P(c)$ and $S(c, \perp_1)$ are due to the atom $U(c)$ in MV and to the tgd σ_U . At the

same time, the atom $P(c)$ is also due to the atom $V(c)$ in MV and to the $\text{tgd } \sigma_V$.

As described in Appendix J, toward finding all the certain answers to the query Q w.r.t. the materialized-view setting $\mathcal{M}\Sigma$, we now chase the instance J_0 , using both the set $\Sigma = \{\sigma\}$ in $\mathcal{M}\Sigma$, as well as the dependencies $\tau_{U(c)}$ and $\tau_{V(c)}$, as follows. (The two latter dependencies are generated by the view-verified data-exchange approach from the inputs \mathcal{V} and MV .)

$$\tau_{U(c)} : P(X) \wedge S(X, Y) \rightarrow X = c.$$

$$\tau_{V(c)} : P(X) \rightarrow X = c.$$

We do three stages of the chase of the instance J_0 with σ , $\tau_{U(c)}$, and $\tau_{V(c)}$. In Stages I and III, we perform the chase steps with the MV -induced dependencies $\tau_{U(c)}$ and $\tau_{V(c)}$, and in Stage II, we chase the instance with the input dependency σ on the schema \mathbf{P} .

Stage I: A chase step of the instance J_0 with the MV -induced dependencies $\tau_{U(c)}$ and $\tau_{V(c)}$ leaves the instance J_0 unchanged. To indicate that we have performed this stage of the chase, we rename J_0 into J_1 :

$$J_1 = \{P(c), S(c, \perp_1)\}.$$

The MV -induced dependencies $\tau_{U(c)}$ and $\tau_{V(c)}$ do not apply to the instance J_1 .

Stage II: We now chase the instance J_1 with the $\text{tgd } \sigma$ on the schema \mathbf{P} . The application adds to the instance J_1 the atom $P(\perp_1)$. We call the resulting instance J_2 :

$$J_2 = \{P(c), S(c, \perp_1), P(\perp_1)\}.$$

The $\text{tgd } \sigma$ does not apply to the instance J_2 .

Note that if we stop after this Stage II, the set of answers without nulls to the query Q on the instance J_2 is $Q(J_2) = \emptyset$. However, we observe that the instance J_2 does not satisfy the MV -induced dependency $\tau_{V(c)}$. We can then do Stage III of the chase, by applying the dependencies $\tau_{U(c)}$ and $\tau_{V(c)}$ to the instance J_2 . An application of $\tau_{V(c)}$ in a chase step to J_2 binds the null \perp_1 , in the atoms $P(\perp_1)$ and $S(c, \perp_1)$, to the constant c . We call the resulting instance J_3 :

$$J_3 = \{P(c), S(c, c)\}.$$

The set of answers without nulls to the query Q on the instance J_3 is $Q(J_3) = \{c\}$. By the results reported in Appendix J, this set is a correct set of certain answers to Q w.r.t. the given materialized-view setting $\mathcal{M}\Sigma$. We can see that by not applying Stage III in the chase, we would have missed the certain answer (c) to the query Q w.r.t. $\mathcal{M}\Sigma$. \square

M. THE CERTAIN-QUERY-ANSWER PROBLEM IS Π_2^P COMPLETE FOR CONJUNCTIVE WEAKLY ACYCLIC INPUT INSTANCES

In this appendix we prove that the certain-query-answer problem, for a query and ground tuple w.r.t. a materialized-view-setting, is Π_2^P complete for CQ weakly acyclic input instances $(\mathcal{M}\Sigma, Q, \bar{t})$. We say that a triple $(\mathcal{M}\Sigma, Q, \bar{t})$, with $\mathcal{M}\Sigma$ a materialized-view setting, Q a

query, and \bar{t} a ground tuple, is a *CQ weakly acyclic input instance* if and only if $\mathcal{M}\Sigma$ is CQ weakly acyclic and Q is a CQ query.

In the complexity measure used throughout this appendix, we assume, in a natural extension of the complexity setting introduced in [31] (see Section J.3 for the detailed discussion), that all elements of $\mathcal{M}\Sigma$ except MV are fixed, and that Q is not fixed. That is, the size of a given input instance $(\mathcal{M}\Sigma, Q, \bar{t})$ is the size of its set of view answers MV and of its query Q , with the remaining elements of $(\mathcal{M}\Sigma, Q, \bar{t})$ being fixed. Note that in all input instances $(\mathcal{M}\Sigma, Q, \bar{t})$ in which \bar{t} could be a certain answer to Q w.r.t. $\mathcal{M}\Sigma$, the size of the ground input tuple \bar{t} must be linear in the size of Q ; more precisely, the size of \bar{t} must be the arity of the query Q . Thus, in this appendix we restrict our consideration to the problem-input triples that satisfy this property. That is, in all of the results in this appendix we assume that, in all the given input instances $(\mathcal{M}\Sigma, Q, \bar{t})$, we have that: Q is a k -ary CQ query for some $k \geq 0$; \bar{t} is a k -ary ground tuple; and the size of the CQ weakly acyclic instance $(\mathcal{M}\Sigma, Q, \bar{t})$ is the size of the set of answers MV in $\mathcal{M}\Sigma$ and of the query Q .

We first observe that the problem is in Π_2^P .

PROPOSITION M.1. *The certain-answer problem for a query and a tuple w.r.t. a materialized-view setting is in Π_2^P for CQ weakly acyclic input instances.* \square

PROOF. Given a CQ weakly acyclic input instance $(\mathcal{M}\Sigma, Q, \bar{t})$, we show how to ascertain that the ground tuple \bar{t} is not a certain answer to the query Q w.r.t. the setting $\mathcal{M}\Sigma$. Observe first that if this is the case, then, by soundness of view-verified data exchange (see Section J.3), there must be a view-verified universal solution, J , for $\mathcal{M}\Sigma$ such that \bar{t} is not an answer to Q on the instance J . We can thus:

- (1) Guess a view-verified universal solution J for $\mathcal{M}\Sigma$, and then
- (2) Verify that there is no valuation from the query $Q(\bar{t})$ to J ; here, $Q(\bar{t})$ is the result of applying to the query Q the homomorphism¹² μ , such that:
 - (i) μ maps the head vector \bar{X} of Q to \bar{t} , and
 - (ii) μ is the identity mapping on each term that occurs in Q but not in its head vector \bar{X} .

By Proposition J.1, the step (1) that generates the instance J can be done in polynomial space in the size of the set MV in $\mathcal{M}\Sigma$. Further, step (2) can be done using an NP -oracle for Q and J , as the size of each valuation from $Q(\bar{t})$ to J must be polynomial in the size of Q and J (it is, in fact, linear in the size of Q). \square

We now provide a Π_2^P hardness result, even for the special case of CQ weakly acyclic inputs with $\Sigma = \emptyset$.

THEOREM M.1. *The certain-answer problem for a query and a tuple w.r.t. a materialized-view setting is Π_2^P hard for CQ input instances $(\mathcal{M}\Sigma, Q, \bar{t})$ in which $\Sigma = \emptyset$ in the input materialized-view setting $\mathcal{M}\Sigma$.* \square

¹²It is easy to verify that if a homomorphism μ specified by (i)-(ii) does not exist, then \bar{t} cannot be a certain answer to Q w.r.t. $\mathcal{M}\Sigma$.

Before providing a proof of Theorem M.1, we observe that as an immediate corollary of Theorem M.1 and of Proposition M.1 we obtain the main result of this appendix, a Π_2^P -completeness result for the certain-answer problem for a query and a tuple w.r.t. a materialized-view setting, for the case of CQ weakly acyclic input instances:

THEOREM M.2. *The certain-answer problem for a query and a tuple w.r.t. a materialized-view setting is Π_2^P complete for CQ weakly acyclic input instances.* \square

In the remainder of this appendix, we provide a proof of Theorem M.1. As a summary, the result of Theorem M.1 is by reduction from the $\forall\exists$ -CNF problem, which is known to be Π_2^P complete [26]. We start off from the reduction that was used by Millstein and colleagues in [21] for the problem of query containment for data-integration systems. We modify the reduction of [21] in the spirit that is similar to the modification of that reduction (of [21]) as suggested in [31]. (Recall that the full version of [31], including any of its proofs, has never been published.) The goal of our modification is to comply with our assumptions about the input size, specifically with the assumption that the input view definitions are fixed. (In [21] it is assumed that both the queries and the view definitions can vary.)

PROOF. (Theorem M.1) In this proof, we build on the constructions from the proof of Theorem 3.3 in [21]; that result of [21] states Π_2^P hardness for a subclass of the problem of query containment for data-integration systems. The reason that we modify the reduction of [21] is that we need to comply with our assumptions about the size of our input instances \mathcal{MS} , specifically with the assumption that the input view definitions are fixed. (In [21] it is assumed that both the queries and the view definitions can vary.) Thus, our variation on the reduction of [21] is similar in spirit to the modification suggested in [31].

Similarly to the reduction in [21], we reduce the $\forall\exists$ -CNF problem, known to be Π_2^P complete [26], to our problem. The $\forall\exists$ -CNF problem is defined as follows: Given a 3-CNF propositional formula F with variables \bar{X} and \bar{Y} , is it the case that for each truth assignment to \bar{Y} , there exists a truth assignment to \bar{X} that satisfies F ? Here, we denote by \bar{X} the set of n variables X_1, \dots, X_n , for some $n \geq 0$, and we denote by \bar{Y} the set of m variables Y_1, \dots, Y_m , for some $m \geq 1$.

The reduction is as follows. Suppose we are given a 3-CNF formula F , with variables

$$\bar{Z} = \{X_1, \dots, X_n\} \cup \{Y_1, \dots, Y_m\}.$$

The formula F has clauses $\bar{C} = \{C_1, \dots, C_l\}$. Clause C_i contains the three variables (either positive or negated) $Z_{i,1}$, $Z_{i,2}$, and $Z_{i,3}$; each of the three variables is an element of the set \bar{Z} .

For the input formula F , we begin building the corresponding (CQ weakly acyclic) instance $(\mathcal{MS}, Q, \bar{t})$ of the certain-answer problem for a query and a tuple w.r.t. a materialized-view setting. In each such instance $(\mathcal{MS}, Q, \bar{t})$, the query Q will be Boolean, hence \bar{t} will be the empty tuple. The setting \mathcal{MS} that we will construct is as usual a quadruple of the form $(\mathbf{P}, \Sigma, \mathcal{V}, MV)$, always with $\Sigma = \emptyset$. We show below how

to construct each of \mathbf{P} , \mathcal{V} , MV , and Q for the input formula F .

The schema $\mathbf{P} = \{P, R, S\}$ that we construct for the input formula F uses three relation symbols: R of arity $k_R = 4$, and two binary relation symbols P and S . Intuitively, for each $i \in [1, l]$ and for the clause C_i in F , in each “relevant” instance of schema \mathbf{P} we will have in R a nonempty set of tuples whose fourth argument is the constant i . Further, for each $j \in [1, m]$ and for the variable Y_j in F , in each “relevant” instance of schema \mathbf{P} we will have in each of P and S a nonempty set of tuples whose second argument is the constant j .

We now define the set of views $\mathcal{V} = \{U, V, W\}$ in the materialized-view setting \mathcal{MS} that we construct for the given formula F . First, for the clauses \bar{C} in F we introduce the following view V :

$$V(Z_1, Z_2, Z_3, i) \leftarrow R(Z_1, Z_2, Z_3, i).$$

(Here, i is a variable rather than a constant; we use the variable name i in the definition of the view V to mnemonically refer to each clause C_i in \bar{C} as discussed above.) The answer to this view simply mirrors the relation R .

In the set MV of answers to the views in \mathcal{V} that we are constructing for the given formula F , the relation $MV[V]$ records, for each $i \in [1, l]$ and for the clause C_i in F , the seven (out of the total eight possible) satisfying assignments for the clause. (We follow [21] in using 1 for *true* and 0 for *false*.) The fourth argument of each tuple in $MV[V]$ for these seven assignments for C_i is always i .

As a running example, we use the following example from the proof of Theorem 3.3 in [21]: Consider the formula

$$F = (X_1 \vee X_2 \vee Y_1) \wedge (\neg X_1 \vee \neg X_2 \vee \neg Y_2).$$

The seven satisfying assignments to X_1 , X_2 , and Y_1 in the first clause $C_1 = (X_1 \vee X_2 \vee Y_1)$ of F are $(1, 1, 1)$, $(1, 1, 0)$, $(1, 0, 1)$, $(1, 0, 0)$, $(0, 1, 1)$, $(0, 1, 0)$, and $(0, 0, 1)$. For the second clause $C_2 = (\neg X_1 \vee \neg X_2 \vee \neg Y_2)$ of F , the seven satisfying assignments to X_1 , X_2 , and Y_2 are $(0, 0, 0)$, $(0, 0, 1)$, $(0, 1, 0)$, $(0, 1, 1)$, $(1, 0, 0)$, $(1, 0, 1)$, and $(1, 1, 0)$.

By construction of the view V and by our intuition for the relation R , see above, in this running example we construct the relation $MV[V]$ from these fourteen assignments, as follows. First, the seven assignments as above for C_1 are adorned, in the fourth argument of V , by the index 1 of C_1 , as follows: $V(1, 1, 1, 1)$, $V(1, 1, 0, 1)$, $V(1, 0, 1, 1)$, $V(1, 0, 0, 1)$, $V(0, 1, 1, 1)$, $V(0, 1, 0, 1)$, and $V(0, 0, 1, 1)$. Similarly, the seven assignments as above for C_2 get adorned, in the fourth argument of V , by the index 2 of C_2 , as follows: $V(0, 0, 0, 2)$, $V(0, 0, 1, 2)$, $V(0, 1, 0, 2)$, $V(0, 1, 1, 2)$, $V(1, 0, 0, 2)$, $V(1, 0, 1, 2)$, and $V(1, 1, 0, 2)$. These fourteen tuples together constitute the relation $MV[V]$ for the formula F in this running example.

We now return from our running example, to continue to define the views in the set $\mathcal{V} = \{U, V, W\}$ for the formula F . For the $m \geq 1$ variables Y_1, \dots, Y_m in F , we introduce a unary view W :

$$W(j) \leftarrow P(Y_j, j), S(Y_j, j).$$

Here, j is a variable rather than a constant; we use the variable name j in the definition of the view W to

mnemonically refer to each variable Y_j in the formula F as discussed in the beginning of this proof.

The relation $MV[W]$ for the given formula F is $MV[W] = \{(1), (2), \dots, (m)\}$. Intuitively, for each $j \in [1, m]$, the tuple (j) in $MV[W]$ witnesses, in each ground instance I of schema \mathbf{P} such that $\mathcal{V} \Rightarrow_{I, \Sigma} MV$, the presence of a ground P -atom $P(y_j, j)$ and of a ground S -atom $S(y_j, j)$ with the same arguments. (Here, y_j is some constant value.) In our running example, we have that $MV[W] = \{(1), (2)\}$, with one tuple for each of the two variables Y_1 and Y_2 in the given formula F .

Finally, in the general case we define the view U in the set $\mathcal{V} = \{U, V, W\}$ for the given formula F as follows:

$$U(Y, j) \leftarrow S(Y, j).$$

(As in the previous view definitions, j is a variable rather than a constant.) The answer to this view simply mirrors the relation S .

Now in the set MV that we are constructing, the relation $MV[U]$ for U provides the two possible truth assignments, 1 and 0, to each variable among Y_1, \dots, Y_m in the set of variables \bar{Y} in the formula F . That is, for each $j \in [1, m]$, the relation $MV[U]$ has exactly two tuples, $U(1, j)$ and $U(0, j)$. For instance, the relation $MV[U]$ for our running example would have four tuples: $MV[U] = \{U(1, 1), U(0, 1), U(1, 2), U(0, 2)\}$. Here, the first two tuples correspond to the two possible truth assignments, 1 and 0, to the variable Y_1 in the formula F in the example. Similarly, the last two tuples in $MV[U]$ correspond to the two possible truth assignments, 1 and 0, to the variable Y_2 in the formula F in the example.

For the general case of the formula F , the above construction generates, for a given F , both the ground tuple $\bar{t} = ()$ and the elements \mathbf{P} , $\Sigma (= \emptyset)$, \mathcal{V} , and MV in the materialized-view setting $\mathcal{M}\Sigma$ that we are producing for F . To complete the construction of the input instance $(\mathcal{M}\Sigma, Q, \bar{t})$ for the given F , we now specify the CQ query Q :

$$Q() \leftarrow \bigwedge_{j=1}^m P(Y_j, j) \bigwedge_{i=1}^l R(Z_{i,1}, Z_{i,2}, Z_{i,3}, i).$$

Intuitively, the Boolean query Q has a separate subgoal, $P(Y_j, j)$, for each $j \in [1, m]$, that is for each Y_j among the variables Y_1, \dots, Y_m of the input formula F . The query Q also has a separate subgoal, $R(Z_{i,1}, Z_{i,2}, Z_{i,3}, i)$, for each $i \in [1, l]$, that is for each $C_i(Z_{i,1}, Z_{i,2}, Z_{i,3})$ among the clauses C_1, \dots, C_l of the input formula F . By design, Q uses in each of its R -subgoals all the variables of the form $Z_{i,k}$ in the same way as they are used in the corresponding clause C_i in the formula F . (Recall that for each variable of the form $Z_{i,k}$ in the clauses of F , this variable is either among the variables \bar{X} of F or among the variables \bar{Y} of F .) In addition, also by design of the query Q , for each variable Y_j among Y_1, \dots, Y_m in the formula F , the *same* variable name Y_j is used in the P -subgoal of Q with j the value of the second argument of the subgoal. It follows that for each variable that is used as the first argument of some P -subgoal of the query Q , this same variable must occur in the conjunction $\bigwedge_{i=1}^l R(Z_{i,1}, Z_{i,2}, Z_{i,3}, i)$ in the body of Q .

As an illustration, the query Q for the formula F in our running example is as follows:

$$Q() \leftarrow P(Y_1, 1), P(Y_2, 2), R(X_1, X_2, Y_1, 1), R(X_1, X_2, Y_2, 2).$$

We have completed the construction of the instance $(\mathcal{M}\Sigma, Q, \bar{t})$ for each 3-*CNF* propositional formula F . By construction, the instance $(\mathcal{M}\Sigma, Q, \bar{t})$ is CQ weakly acyclic and has $\Sigma = \emptyset$ and $\bar{t} = ()$. Further, each of \mathbf{P} , Σ , and \mathcal{V} , in the materialized-view setting $\mathcal{M}\Sigma$ constructed for the input formula F , does not depend on F ; thus, both the formulation and the size of each of \mathbf{P} , Σ , and \mathcal{V} are fixed across all the input formulae F . In contrast, the size of each of Q and MV in the instance $(\mathcal{M}\Sigma, Q, \bar{t})$ is linear in the size of the input formula F . As a result, the overall size of the instance $(\mathcal{M}\Sigma, Q, \bar{t})$ is polynomial (linear) in the size of the input formula F .

It turns out that for each formula F , the materialized-view setting $\mathcal{M}\Sigma$ that we construct for F is a valid setting by definition. Indeed, for each such F and for the corresponding setting $\mathcal{M}\Sigma = (\mathbf{P}, \Sigma, \mathcal{V}, MV)$, there exists a ground instance $I^{(1,1,\dots,1)}(F)$ of schema \mathbf{P} such that $\mathcal{V} \Rightarrow_{I^{(1,1,\dots,1)}(F), \Sigma} MV$, as follows. (Intuitively, the m -tuple $(1, 1, \dots, 1)$ in the name of this instance $I^{(1,1,\dots,1)}(F)$ refers to the fact that this instance represents an assignment of the truth value 1 to each of the m variables Y_1, \dots, Y_m of the input formula F .) $I^{(1,1,\dots,1)}(F)$ is the union of the instance $I_{(P,S)}^{(1,1,\dots,1)}(F)$, as specified below, with the instance $I_{(R)}^{(1,1,\dots,1)}(F)$ that has the same set of tuples for the relation R as the instance MV has in the relation for V (see the specification of $MV[V]$ above). As an illustration, in our running example, the instance $I_{(R)}^{(1,1)}(F)$ has the same fourteen tuples as we saw above in the relation $MV[V]$, except that these same tuples are now in the relation R in $I^{(1,1)}(F)$:

$$\begin{aligned} I_{(R)}^{(1,1)}(F) = \{ & R(1, 1, 1, 1), R(1, 1, 0, 1), R(1, 0, 1, 1), \\ & R(1, 0, 0, 1), R(0, 1, 1, 1), R(0, 1, 0, 1), \\ & R(0, 0, 1, 1), R(0, 0, 0, 2), R(0, 0, 1, 2), \\ & R(0, 1, 0, 2), R(0, 1, 1, 2), R(1, 0, 0, 2), \\ & R(1, 0, 1, 2), R(1, 1, 0, 2)\}. \end{aligned}$$

For the general case, the instance $I_{(P,S)}^{(1,1,\dots,1)}(F)$ is of the following form:

$$\begin{aligned} I_{(P,S)}^{(1,1,\dots,1)}(F) = \{ & P(1, 1), P(1, 2), \dots, P(1, m), S(1, 1), \\ & S(0, 1), S(1, 2), S(0, 2), \dots, S(1, m), S(0, m)\}. \end{aligned}$$

As an illustration, for our running example, the instance $I_{(P,S)}^{(1,1)}(F)$ is as follows:

$$\begin{aligned} I_{(P,S)}^{(1,1)}(F) = \{ & P(1, 1), P(1, 2), S(1, 1), S(0, 1), \\ & S(1, 2), S(0, 2)\}. \end{aligned}$$

The entire instance $I^{(1,1)}(F)$ for our running example is the union of the instances $I_{(P,S)}^{(1,1)}(F)$ and $I_{(R)}^{(1,1)}(F)$ as given above.

In the general case, intuitively, the tuples in the relation S in $I_{(P,S)}^{(1,1,\dots,1)}(F)$ mirror the instance $MV[U]$, by definition of the view U and of $MV[U]$. The tuples $P(1, 1), P(1, 2), \dots, P(1, m)$ in $I_{(P,S)}^{(1,1,\dots,1)}(F)$ give us a key part of this proof, by representing a particular assignment of the truth values 1 and 0, one value to each

Y_j ($j \in [1, m]$) among the variables Y_1, \dots, Y_m of the formula F . The particular assignment of the truth values 1 and 0 to the variables Y of F that is represented in the instance $I^{(1,1,\dots,1)}(F)$ is the assignment of the truth value 1 to each of the m variables. We represent this fact in the name of the instance $I^{(1,1,\dots,1)}(F)$, by using this assignment as an m -tuple $(1, 1, \dots, 1)$ in the superscript in the name.

It is straightforward to verify that the ground instance $I^{(1,1,\dots,1)}(F)$ satisfies $\mathcal{V} \Rightarrow_{I^{(1,1,\dots,1)}(F), \Sigma} MV$. Further, it is straightforward to verify that there exist $2^m - 1$ more ground instances of schema \mathbf{P} , as follows. Each such instance, denote it for now by J , differs from the instance $I^{(1,1,\dots,1)}(F)$ only in whether the first argument of one or more P -tuple(s) in J is the value 0, instead of 1 as it is in $I^{(1,1,\dots,1)}(F)$. It is straightforward to verify that for each such instance J , we have that $\mathcal{V} \Rightarrow_{J, \Sigma} MV$ in the context of the setting $\mathcal{M}\Sigma$ that we have constructed for the given formula F . Clearly, the total number of such instances J , including the instance $I^{(1,1,\dots,1)}(F)$, is 2^m , as the value 1 of the first argument of $P(1, j)$ in $I^{(1,1,\dots,1)}(F)$ can be flipped to 0 independently for each $j \in [1, m]$.

For each instance J constructed as above, we name the instance using the same notation as for the instance $I^{(1,1,\dots,1)}(F)$, by adorning the name $I(F)$ with an m -tuple (a_1, a_2, \dots, a_m) in the superscript, where a_j for each $j \in [1, m]$ is either 1 or 0, and (a_j) is precisely the first argument of the P -atom in the instance such that this P -atom has j as its second argument. For example, the instance $I^{(0,0,\dots,0)}(F)$ is the instance that differs from the instance $I^{(1,1,\dots,1)}(F)$ only in that the first argument of each P -tuple in J is the value 0, instead of 1 as it is in $I^{(1,1,\dots,1)}(F)$.

To the 2^m instances of the form $I^{(a_1, a_2, \dots, a_m)}(F)$ as defined above, we refer collectively as *the core instances (of schema \mathbf{P}) for the input formula F* . In addition to these core instances, there is an infinite number of other ground instances of schema \mathbf{P} , where for each instance, I , we have that $\mathcal{V} \Rightarrow_{I, \Sigma} MV$ in the context of the setting $\mathcal{M}\Sigma$ that we have constructed for the given formula F . By definition of $\mathcal{M}\Sigma$, each such instance I can be obtained by unioning one of the core instances for F , call the instance K , with a finite set of ground atoms of the form $P(c, d)$, where either (i) d is a constant that is not in the set $[1, m]$, or (ii) d is in the set $[1, m]$, while c is a constant distinct from the value g in the atom $P(g, d)$ in the “core part” K of the instance I .

For each ground instance I that satisfies $\mathcal{V} \Rightarrow_{I, \Sigma} MV$ and by definition of the query Q that we have constructed for the given formula F , we obtain the following useful observations.

LEMMA M.1. *Given a 3-CNF propositional formula F and the instance $(\mathcal{M}\Sigma, Q, \bar{t})$ for F . Then, for all ground instances I of schema \mathbf{P} such that $\mathcal{V} \Rightarrow_{I, \Sigma} MV$ in the context of the instance $(\mathcal{M}\Sigma, Q, \bar{t})$, the relation R is the same in all the instances I , and the relation S is also the same in all the instances I .* \square

This result is immediate from the definitions of the views U and V . The result of Lemma M.1 implies that in each ground instance I of schema \mathbf{P} such that $\mathcal{V} \Rightarrow_{I, \Sigma} MV$, (i) the relation R is the same in I as the relation

R in the fixed instance $I^{(1,1,\dots,1)}(F)$ defined above, and (ii) the relation S is the same in I as the relation S in $I^{(1,1,\dots,1)}(F)$.

LEMMA M.2. *Given a 3-CNF propositional formula F and the instance $(\mathcal{M}\Sigma, Q, \bar{t})$ for F . Then, for each ground instance I of schema \mathbf{P} such that $\mathcal{V} \Rightarrow_{I, \Sigma} MV$ in the context of the instance $(\mathcal{M}\Sigma, Q, \bar{t})$, the relation P in I has for each $j \in [1, m]$ an atom of the form $P(e_j, j)$, with the constant $e_j \in \{1, 0\}$.* \square

PROOF. This result is immediate from the definitions of the views U and W and from the specifications of the relations $MV[U]$ and $MV[W]$ in $\mathcal{M}\Sigma$ as constructed for the given formula F . Indeed, recall that $MV[W] = \{(1), (2), \dots, (m)\}$. By definition of the view W , for each $j \in [1, m]$ we have that the tuple (j) in $MV[W]$ witnesses, in each ground instance I of schema \mathbf{P} such that $\mathcal{V} \Rightarrow_{I, \Sigma} MV$, the presence of a ground P -atom $P(e_j, j)$ and of a ground S -atom $S(e_j, j)$ with the same arguments. Now consider an arbitrary ground atom $S(d, f)$ in any ground instance I of schema \mathbf{P} such that $\mathcal{V} \Rightarrow_{I, \Sigma} MV$. By definition of the view U and by the contents of the relation $MV[U]$, the value d in this atom $S(d, f)$ must be one of 1 and 0, and the value f in $S(d, f)$ must belong to the set $[1, m]$. Thus, via the definition of the view W and the contents of the relation $MV[W]$ as discussed above, we obtain that each ground atom of the form $P(e_j, j)$ as above must have its value e_j restricted to one of 1 and 0. The claim of the lemma follows. \square

LEMMA M.3. *Given a 3-CNF propositional formula F and the instance $(\mathcal{M}\Sigma, Q, \bar{t})$ for F . Let I be a ground instance of schema \mathbf{P} such that $\mathcal{V} \Rightarrow_{I, \Sigma} MV$ in the context of the instance $(\mathcal{M}\Sigma, Q, \bar{t})$. Then there exists a core instance K for F such that there is an identity homomorphism from K to I .* \square

LEMMA M.4. *Given a 3-CNF propositional formula F and the instance $(\mathcal{M}\Sigma, Q, \bar{t})$ for F . Let I be a ground instance of schema \mathbf{P} such that $\mathcal{V} \Rightarrow_{I, \Sigma} MV$ in the context of the instance $(\mathcal{M}\Sigma, Q, \bar{t})$. Then for each valuation, μ , from Q to I , the image of all the P -subgoals of Q under μ is the relation for P in one of the core instances for F .* \square

(Toward the proof of Lemma M.4, recall that by design of the query Q , the first argument of each P -subgoal of the query Q must also occur as one of the first three arguments of at least one R -subgoal of Q .)

By the above lemmata and by the structure of all the ground instances I of schema \mathbf{P} such that $\mathcal{V} \Rightarrow_{I, \Sigma} MV$, it must be that, in the context of the instance $(\mathcal{M}\Sigma, Q, \bar{t})$:

(*) For each I such that $\mathcal{V} \Rightarrow_{I, \Sigma} MV$ and for each valuation, μ , from the query Q to I , the image $\mu(\text{body}(Q))$ of the body of the query Q under μ is a subset of one of the core instances for F .

Specifically, by construction of the query Q , for any such core instance K for F , all the ground atoms in the relation P in K must be present in the set $\mu(\text{body}(Q))$. (Recall that in each core instance, K , for the input formula F , the relation P intuitively represents exactly

one specific assignment of values 1 and 0 to the $m \geq 1$ variables \bar{Y} of the formula F . Further, each specific assignment of values 1 and 0 to the $m \geq 1$ variables \bar{Y} of the formula F is represented by a separate core instance for F .)

As an illustration, consider the query Q of our running example and the instance $I^{(1,1)}(F)$ of schema \mathbf{P} for that example. (Both the query Q and the instance $I^{(1,1)}(F)$ for this running example have already been given in this proof.) Consider a mapping $\mu = \{X_1 \rightarrow 1, X_2 \rightarrow 0, Y_1 \rightarrow 1, Y_2 \rightarrow 1\}$. We can show that μ is a valuation from the query Q to the instance $I^{(1,1)}(F)$. The image $\mu(\text{body}_{(Q)})$ of the body of the query Q under the valuation μ includes all the ground atoms in the relation P in the instance $I^{(1,1)}(F)$, that is both atoms $P(1, 1)$ and $P(1, 2)$ in $I^{(1,1)}(F)$.

We now proceed to show that for the input formula F and for the corresponding instance $(\mathcal{M}\Sigma, Q, \bar{t})$ constructed for F as above, the following two statements are equivalent:

- (I) For each assignment of truth values 1 and 0 to the variables \bar{Y} in the formula F , there exists an assignment of truth values 1 and 0 to the variables \bar{X} in F such that F is true under these assignments; and
- (II) The tuple $()$ is an answer to the query Q on all the ground instances I of schema \mathbf{P} such that $\mathcal{V} \Rightarrow_{I, \Sigma} MV$ in the context of the instance $(\mathcal{M}\Sigma, Q, \bar{t})$.

Note that by the definition in Section 4.1, the statement (II) says that the tuple $()$ is a certain-answer tuple for the query Q w.r.t. the setting $\mathcal{M}\Sigma$. Thus, once we show the equivalence of the statements (I) and (II), our proof of Π_2^P hardness of the certain-query-answer problem for CQ weakly acyclic inputs with $\Sigma = \emptyset$ will be complete.

We begin the proof of the equivalence of the statements (I) and (II) by making the following observation. Denote by $\mathcal{R}_{(Q)}$ the conjunction of the R -subgoals of the query Q in the instance $(\mathcal{M}\Sigma, Q, \bar{t})$ for the given formula F . Further, denote by $\mathcal{P}_{(Q)}$ the conjunction of the P -subgoals of the query Q in the instance $(\mathcal{M}\Sigma, Q, \bar{t})$. Consider any ground instance I of schema \mathbf{P} such that $\mathcal{V} \Rightarrow_{I, \Sigma} MV$ in the context of the instance $(\mathcal{M}\Sigma, Q, \bar{t})$. Let ν be any mapping of the set \bar{Z} of the variables (\bar{X} and \bar{Y}) of the formula F into the set $\{0, 1\}$. Similarly to the argument in [3] (as also used in the proof of Theorem 3.3 in [21]), we can show that any such mapping ν is a satisfying assignment for the formula F if and only if $\nu(\mathcal{R}_{(Q)})$ is a subset of the instance I . By Lemma M.1, we have that any such mapping ν is a satisfying assignment for the formula F if and only if $\nu(\mathcal{R}_{(Q)})$ is a subset of *each* ground instance J of schema \mathbf{P} such that $\mathcal{V} \Rightarrow_{J, \Sigma} MV$ in the context of the instance $(\mathcal{M}\Sigma, Q, \bar{t})$. From the above reasoning and from Lemmas M.2–M.3, we obtain the following result of Lemma M.5.

We first introduce some notation. In the remainder of the proof of Theorem M.1, let $\mu_{(\bar{Y})}^{(a_1, \dots, a_m)}$, with $a_j \in \{0, 1\}$ for each $j \in [1, m]$, denote the mapping from the set of variables \bar{Y} of the formula F to the set $\{0, 1\}$, such that $\mu_{(\bar{Y})}^{(a_1, \dots, a_m)}(Y_j) = a_j$ for each $j \in [1, m]$. Further, let $\mu_{(\bar{X})}$ denote a mapping from the set of variables \bar{X}

of the formula F to the set $\{0, 1\}$.

LEMMA M.5. *Given a 3-CNF formula F with $m \geq 1$ variables \bar{Y} and with $n \geq 0$ variables \bar{X} , let (a_1, \dots, a_m) be an arbitrary m -tuple such that $a_j \in \{0, 1\}$ for each $j \in [1, m]$. Let $\mu_{(\bar{X})}$ be an arbitrary mapping from the set of variables \bar{X} of the formula F to the set $\{0, 1\}$. Let $(\mathcal{M}\Sigma, Q, \bar{t})$ be the instance that we have constructed for the formula F as above. Finally, let I be a ground instance of the schema \mathbf{P} , such that $\mathcal{V} \Rightarrow_{I, \Sigma} MV$ in the context of the instance $(\mathcal{M}\Sigma, Q, \bar{t})$, and such that the set $\{P(a_1, 1), \dots, P(a_m, m)\}$ is a subset of the instance I .*

Then the following two statements are equivalent:

- *The assignment $\mu_{(\bar{Y})}^{(a_1, \dots, a_m)} \cup \mu_{(\bar{X})}$ of the variables of the formula F to elements of the set $\{0, 1\}$ is a satisfying assignment for the formula F ; and*
- *The empty tuple $()$ is in the answer to the query Q on the instance I due to the valuation $\mu_{(\bar{Y})}^{(a_1, \dots, a_m)} \cup \mu_{(\bar{X})}$ from $\text{body}_{(Q)}$ to I .* \square

We are now ready to show the equivalence of the statements (I) and (II) as formulated above.

(I) \rightarrow (II): Suppose that for each m -tuple (a_1, \dots, a_m) , such that $a_j \in \{0, 1\}$ for each $j \in [1, m]$, we have that there exists a mapping $\mu_{(\bar{X})}$ from the set of variables \bar{X} of the formula F to the set $\{0, 1\}$, such that

$$\mu_{(\bar{Y})}^{(a_1, \dots, a_m)} \cup \mu_{(\bar{X})}$$

is a satisfying assignment for the formula F . Fix an arbitrary ground instance I of the schema \mathbf{P} such that $\mathcal{V} \Rightarrow_{I, \Sigma} MV$. Then, by Lemmas M.2 and M.5, the empty tuple $()$ is in the relation $Q(I)$.

(II) \rightarrow (I): Consider the set \mathcal{K} of the 2^m core instances (of schema \mathbf{P}) for the formula F . By construction of the set \mathcal{K} , for each m -tuple (a_1, \dots, a_m) such that $a_j \in \{0, 1\}$ for each $j \in [1, m]$, there exists an instance $K \in \mathcal{K}$ such that the set $\{P(a_1, 1), \dots, P(a_m, m)\}$ is a subset of the instance K , and the relation $K[P]$ has *no other tuples*.

Fix an arbitrary instance $K \in \mathcal{K}$; the relation $K[P] = \{P(a_1, 1), \dots, P(a_m, m)\}$ specifies a particular m -tuple (a_1, \dots, a_m) such that $a_j \in \{0, 1\}$ for each $j \in [1, m]$. By our assumption (II), there exists a mapping $\mu_{(\bar{X})}$ from the set of variables \bar{X} of the formula¹³ F to the set $\{0, 1\}$, such that

$$\mu(K) = \mu_{(\bar{Y})}^{(a_1, \dots, a_m)} \cup \mu_{(\bar{X})}$$

is a valuation from the query Q to the instance K that produces the empty tuple $()$ in the relation $Q(K)$. Thus, by Lemma M.5, the mapping $\mu(K)$ of the variables of the formula F to the set $\{0, 1\}$ is a satisfying assignment for the formula F . The claim of (I) follows from the observation (made above) that for each m -tuple (a_1, \dots, a_m) such that $a_j \in \{0, 1\}$ for each $j \in [1, m]$, there exists an instance $K \in \mathcal{K}$ such that the set $\{P(a_1, 1), \dots, P(a_m, m)\}$ is in the instance K . This completes the proof of Theorem M.1.

¹³Recall that the set $\bar{Z} = \bar{X} \cup \bar{Y}$ is the set of all variables of the formula F , and is also the set of all variables of the query Q in $(\mathcal{M}\Sigma, Q, \bar{t})$.